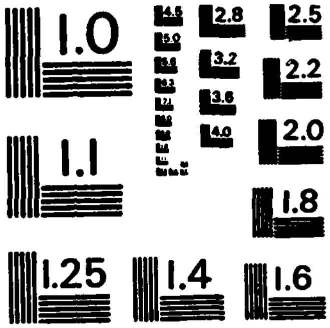
AD-A121 360 AUTOMATION OF QUALITY MEASUREMENT(U) GENERAL ELECTRIC 1/2
CO UUNNYVALE CALIF J A MCCALL ET AL. SEP 82
RADC-TR-82-247 F30602-79-C-0267

UNCLASSIFIED F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

12

RADC-TR-82-247
Final Technical Report
September 1982

AUTOMATION OF QUALITY MEASUREMENT

General Electric Company

James A. McCall
David Markham

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE NY 13441

US ARMY INSTITUTE FOR RESEARCH IN
MANAGEMENT INFORMATION
AND COMPUTER SCIENCES
ATLANTA GA 30332

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-82-247 has been reviewed and is approved for publication.

APPROVED: Joseph Cavano

JOSEPH CAVANO
Project Engineer

APPROVED: Alan R Barnum

ALAN R. BARNUM
Assistant Chief
Command & Control Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-82-247	2. GOVT ACCESSION NO. AD-A121	3. RECIPIENT'S CATALOG NUMBER 360
4. TITLE (and Subtitle) AUTOMATION OF QUALITY MEASUREMENT		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report Sep 79 - Sep 81
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) James A. McCall David Markham		8. CONTRACT OR GRANT NUMBER(s) F30602-79-C-0267
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Electric Company - Western Systems 1277 Orleans Drive Sunnyvale CA 94086		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63728F 25280201
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (COEE) Griffiss AFB NY 13441		12. REPORT DATE September 1982
		13. NUMBER OF PAGES 164
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same as block 11 and: US Army Computer Systems Command/AIRMICS Georgia Institute of Technology Atlanta GA 30332		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Joseph P. Cavano (315) 330-7834

USACSC Project Engineer: Daniel E. Hocking (404) 894-3111

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Quality
Quality Metrics
Software Measurement
Software Tool

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A prototype software system has been developed which allows manual input and provides for automated collection of software metric data, stores the data, and provides processing and reporting to facilitate use of the metric information to monitor and control the quality of a software product. The software system, call the Automated Measurement Tool, processes COBOL source code.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

Section Page

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

PREFACE

This document is a report prepared for the Rome Air Development Center (RADC) and the US Army Computer Systems Command/AIRMICS in support of the Automation of Quality Measurement project. It is the final report (CDRL A003) for the Contract No. F30602-79-C-0267. The purpose of the project was to provide computer programs, supporting documents, and research results related to the effort of measuring certain quality characteristics of software.

This report was prepared by J. McCall and D. Markham. Contributions were made by R. McGindley, M. Hegedus, M. Matsumoto, A. Stone, and M. Stosick.

Technical guidance was provided by Mr. J. Cavano of RADC and supported by Mr. D. Hocking of AIRMICS.

The objective of this study was to establish and demonstrate a method of automating the measurement of significant aspects of software quality. Conceptually, the method of software measurement through metrics provides a mechanism in conjunction with a vigorous development program to provide management a technique to improve the quality of software products.

SECTION 1
INTRODUCTION

1.1 IDENTIFICATION

This document is the Final Report of the research and development tasks associated with the development of the Automated Metrics Tool (AMT). The development and construction of this prototype software tool was provided by GE Western Systems, Sunnyvale, Calif. in compliance with the requirements set forth by Rome Air Development Center (RADC) and the US Army Computer Systems Command/AIRMICS for the Automation of Quality Measurement Project, Contract No. F30602-79-C-0267.

1.2 SCOPE

This document includes a description of the AMT and the results of some parallel research efforts; the use of AMT on itself to derive quality measurements of the tool, the design goals of the AMT, the conversion of AMT from a VAX 11/780 development machine to a Honeywell 6000 series host environment, and the comparison of metric scores across several projects including the AMT. Also included is a description of how the AMT could be used to support a software development activity.

1.3 ORGANIZATION OF DOCUMENT

This section of the document is an introduction to the remainder of the report. The second section provides a brief description of the background and application of metric concepts referencing previously funded RADC and USACSC ,unded research. The third section will describe how the AMT was developed and the motivation for developing it. Section 4 is a description of the AMT. Section 5 describes how we used the AMT during its development to apply metrics. The Final Section suggests future research in metrics and identifies enhanced capabilities that should be considered for the AMT.

1.4 APPLICABLE DOCUMENTS

The following documents include those published and distributed by RADC which are background to this effort and explain in detail the concept and manual application of software quality metrics, as well as those produced as a result of this research task:

McCall, J.A. et al., "Factors In Software Quality". RADC-TR-77-369, June 1977.

McCall, J.A., and Matsumoto, M.T., "Metrics Enhancement Final Report", RADC-TR-80-109, Volume I, April 1980.

McCall, J.A., and Matsumoto, M.T., "Software Quality Measurement Manual", RADC-TR-80-109, Volume II, April 1980.

The applicable documents produced during the course of this research task, identified with their associated CDRL item number:

AMT User's Manual	A012
AMT Training Material	A006
AMT Program Maintenance Manual	A013
AMT Functional Description	A007
AMT Data Requirement Document	A008
AMT Sys./Subsystem Specification	A009
AMT Program Specification	A010
AMT Data Base Description	A011
AMT Test Plan	A015
Test Analysis	A014
AMT Program Maintenance Manual	A013

1.5 EXECUTIVE SUMMARY

The concept behind software quality metrics is to provide software acquisition managers with a mechanism to quantitatively specify and measure the level of quality in a software product. To provide this mechanism, an acquisition manager or a software developer must collect data from the products of the software development process. The actual data items collected are identified in detail in the "Metric Enhancement Final Report", RADC TR-80-109. This raw data is then used to calculate metric values which can be used to assess the quality of the software being produced.

The purpose of the Automated Measurement Tool (AMT) is to provide automated support to the application of the metrics concept. Normally, the metrics data must be collected by hand in a tedious, error-prone, and time-consuming process.

The intent of the AMT is to automatically collect and store the metric data economically and reliably and provide a data base of metrics data to facilitate research and evaluation. This will then allow the acquisition manager to easily collect and analyze the software quality metric values for any given software development using the AMT. The AMT data base and reporting capabilities were used to the extent possible ouring the development of the AMT itself to support application of metrics. This is the first actual contractual application of metrics and the lessons learned from this experience are documented in this report.

The current version of the AMT operates on the Honeywell 6180/GCOS computer system at RADC. It processes COBOL code.

SECTION 2
BACKGROUND OF SOFTWARE METRICS

The basic concepts for the software metrics automatically collected, calculated, and reported by the AMT were derived during the Factors in Software Quality contract, contract number F30602-76-C-0417. A framework for defining metrics [CAVJ78], applying them to all of the products, including documentation, of a software development, and relating them to management goals was developed [MCCJ77I]. A preliminary handbook for an Air Force System Program Office was developed to describe the framework [MCCJ77III]. Initial validation of some of the metrics was performed using command and control software systems written in JOVIAL that had 2-4 years of operation and maintenance data available [MCCJ77II]. During a subsequent research effort, the Metrics Enhancement Contract, contract number F30602-78-C-0216, further validation was conducted using an Army financial management information system written in COBOL that had been transported to two different vendor's computers besides the initial development system and a software support system written in FORTRAN that had been transported to a number of different DEC operating systems as well as a Honeywell 6000 computer system. At the end of this effort, validation of metrics related to the quality factors Reliability, Maintainability, Flexibility, and Portablility was achieved [MCCJ79I]. More importantly, techniques were developed to apply the metrics and derive information during a software development that facilitated identification of potential quality problems, areas needing improvement in standards and conventions, test strategy, and acceptance criteria. An overview of these techniques was provided in a Software Quality Measurement Manual [MCCJ78II]. Currently, software metrics is one of the most widely investigated subject areas in the software research community. Many individuals and organizations are developing metrics related to or extending the metrics developed under the previously mentioned RADC/USACSC funded research and the pioneering work of the others ([HALM77], [MCCT76], [BOEB73], [CHER], [FAGM76], [FOSL76], [HESC77]). The significance of this continued research is not only the refinement of the set of metrics which can be utilized in the framework established in the report, "Factors in Software Quality" [MCCJ77], but also the industry wide experiences being gained in applying and using metrics during software developments.

As more organizations apply metrics, more quantitative information is becoming available about the software being developed. This information supports research in other software engineering disciplines such as software tools and development environments, programming languages, design techniques, and cost estimation.

It is expected that the use of software metrics will become a standard contractual instrument to assure a certain level of quality.

This growing recognition by government organizations, further refinement by the research community, and application experience by a number of organizations makes the introduction of a tool like the AMT timely.

SECTION 3
DEVELOPMENT OF THE AMT

3.1 CONDUCT OF THE PROJECT

The AMT project was conducted in four phases: design, implementation, .esting, and delivery/training. Our motivation and approach to each phase will be discussed in this section.

3.2 NEED FOR AUTOMATION

At the outset of the project, a specific need was envisioned for a family of software tools that would support the specification, measurement, and reporting of software quality metrics. The viability of effective measurement of software quality has been enhanced by the evolution during the past decade of modern programming practices, structured, disciplined development techniques and methodologies, and requirements for more structured, effective documentation.

The actual measurement of software quality is accomplished by applying software metrics (or measurements) to the documentation and source code produced during a software development. These measurements are part of the established model of software quality and through that model can be related to various user-oriented aspects of software quality.

The current set of metrics utilized in the model which is comprised of 11 quality factors has 39 software metrics. Subsets of these 39 metrics can be applied during each phase of the development. The breakdown by phase is:

15 can be applied during requirements
34 can be applied during design
38 can be applied during implementation

To calculate this entire complement of metrics, 296 individual data items have to be collected. Worksheets, described in paragraph 3.2, contain all of the individual data items. A breakdown by phase of the individual data items is:

31	measured during requirements
173	measured during design
92	measured during implementation
296	Total

All of the data items collected during requirements and 102 of the items collected during design are system level measurements which are taken once. However, the remaining 71 items collected during design and the 92 collected during implementation are module level measurements which are taken for each module in the system. Thus, for even a modestly sized system of 100 modules, the number of data items to be collected is:

31 (requirements) + 102 (design) + 100x71 (design)
+100x92 (implementation) = 16,433.

On a particular development project, a subset of these data items relating to the important quality factors to that development would be collected.

To collect this large a number of data items manually from documentation and source code is a very time-consuming, error prone, and thus expensive process.

The time consuming nature of manual inspection of source code impacts one of the important necessary conditions for the quality assurance effort. For project management to make the necessary corrections to meet a specified quality goal, software quality assurance analysts must be able to report discrepancies as soon as possible after the code is ready for inspection.

Metric data also needs to be archived for developmental, life-cycle and research reasons. Software quality assurance analysts need to have an overall view of the metric scores. Researchers need to have a historical record across projects for comparative purposes. This historical record can most easily be kept, updated, and transmitted if it is in machine readable form. This data base can be used in conjunction with other information such as trouble reports, maintenance records, or cost data to determine if the metric data correlates with parallel or past events. If an environment is calibrated through experience, predictions could then be made. All of these capabilities can be realized if the metric data is placed in a data base.

An important ingredient in an effective software quality assurance effort is to operate in such a fashion that measurement does not interfere significantly with development, test, or other procedures which are critical in production. If metric analysis is done by machines the possibility arises to move a majority of the quality assurance activity off-line of the development activity.

For these three reasons, the tedious nature of examining source, the need for storage and retrival of metric data, and the off-line nature of SQA activity, automation of metric applications is important to software quality metric implementation.

The goals of this project were to make this process more timely, reliable, and economic. Automating the collection and reporting provides these three goals. In addition, the data is maintained in a data base for use as a quality assurance management information system, as an historical record of the development project, and as a repository for researchers to investigate combinations of data items to form new metrics.

The prototype version of the AMT, which is described in more detail in Section 4 of this report and referenced project documents, was developed to demonstrate the effectiveness of these concepts. 25 module level measurements are automatically collected. Thus, of the 16,433 measurements that could potentially be required in our previous example, the AMT automates collection of 2500. The AMT data base accommodates the total complement of data items. The measurements not taken automatically have to be entered into the data base manually.

The 25 data items collected automatically represent 27% of the 92 data items measured during implementation. No attempt was made to automatically measure any data items related to requirements and design. The reasons are discussed in the next paragraph.

3.3 OPERATIONAL CONCEPT

The need for automated support of the application of the metrics is tempered by the fact that the metrics we have defined in previously funded RADC and USACSC projects not only relate to source code but also to material, specifically documentation, normally available during the requirements and design phases of a software development. Because, at the present time, there are few formal requirements or design specification languages widely accepted or used throughout the software industry and the material produced during these phases is not produced in machine readable form, this material does not lend itself to automated measurement. However, we see a trend toward more use of formal requirements and design representations such as SREM, PSL/PSA, PDL and automated analysis tools relating to them. This trend will allow expansion of the AMT in terms of automated collection of metric information during these early stages.

The AMT is a system that is used during all phases of a software development process: requirements specification, design specification, implementation, integration and test, and operation. The capabilities provided by AMT are: automatic collection of specified metric data from machine-readable materials, facilitation of collection and entry of other metrics data, storage and retrieval of metrics data, and generation of different reports for use in tracking software quality. These capabilities are used by four different types of users: a customer or user of the software system being developed, the software project's quality assurance analyst, the software development project manager, and a software quality researcher.

With these constructs in mind, the automated measurement of software quality is designed to work in the following way:

(1) At the beginning of a project the quality goals of the project are stated and desired metric values are determined.

(2) At the conclusion of the requirements phase, worksheet #1 is manually completed and the data is manually entered into the AMT's data base.

(3) When the preliminary design is completed, worksheet #2a is manually completed and entered into the data base.

(4) As various detail designs are completed, worksheets #2b are manually entered for each module. At the conclusion of the design phase an update to #2a is also manually placed in the data base. Steps 2, 3, and 4 utilize the Requirements Specification, Preliminary and Detailed Design documents, test plans, preliminary users manual, and other material normally available during the requirements and design phases of a project.

(5) During implementation, as soon as COBOL source code is placed under configuration management, the source code is measured by the quality measurement tool and this data is also entered into the data base. Additional data, identified on worksheet #3 is manually entered. This application timing is depicted in Figure 3.3-1.

(6) Various updates may be made as required.

(7) At delivery of the software product, it is anticipated that all metric information would be updated to reflect the current state of the code and documentation.

(8) At each stage of application, a number of reports would be generated to provide the appropriate information to various personnel involved in the development.

One primary purpose of the AMT is to calculate the metric scores from the data input from the worksheets. These scores can then be compared with desired scores. This operational concept is depicted in Figure 3.3-2 and is compatible with the methodology described in the Software Quality Measurement Manual [MCCJ79]. Additional helpful information is available to support various decisions, quality assurance activities, and testing activities. Figure 3.3-3 identifies the support provided. This support is described further in Section 4 where each report is defined.

The potential of the software metric concepts can be realized by their inclusion in software quality assurance programs. Their impact on a quality assurance program is to provide a more disciplined, objective approach to quality assurance and to provide a mechanism for taking a life cycle viewpoint of software quality. The benefits derived from their application are realized in life cycle cost reduction.

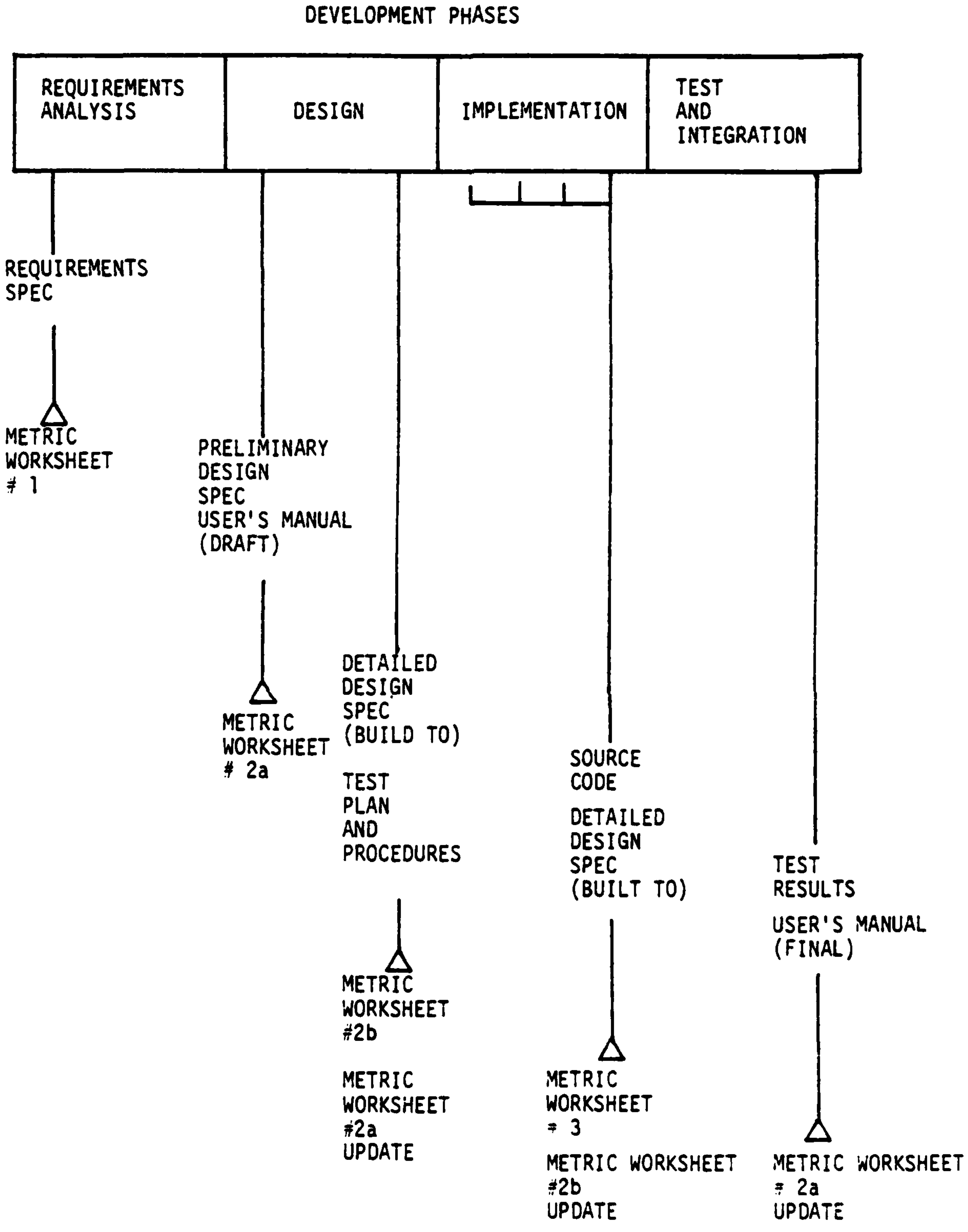


Figure 3.3-1 Application of the Metric Worksheets

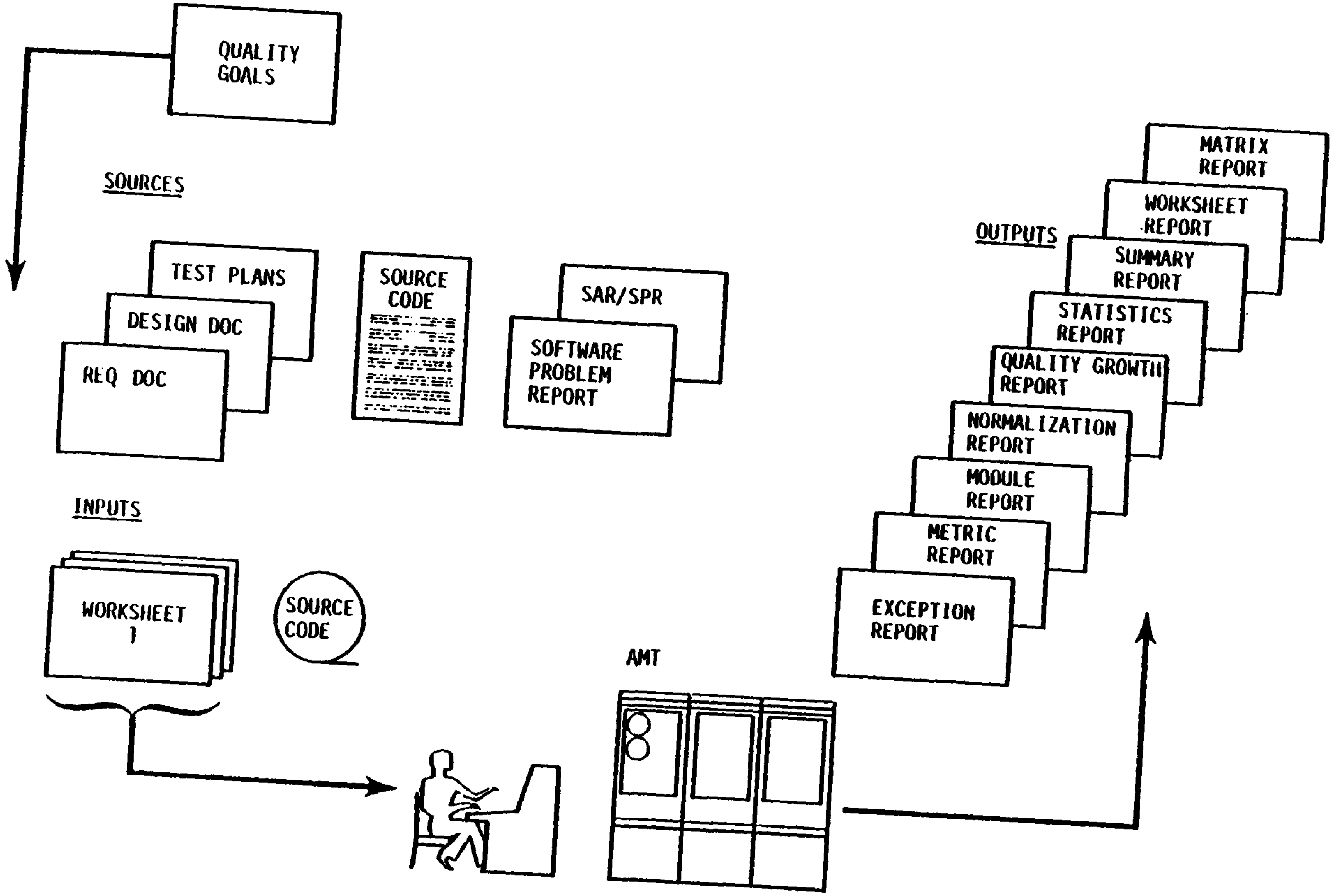


Figure 3.3-2 AMT Operational Concept

PERSONNEL	SUPPORT	REPORT TYPE
Program Management	Progress with Respect to Quality Goals	Normalization Quality Growth Matrix Statistics
Developers	Standard Enforcement Design Decisions	Metric Summary
Quality Assurance	Standards Enforcement Compliance with Quality Goals Problem Identification	Metric Quality Growth Normalization Matrix Module
Test	Test Strategy Test Emphasis Test Effort	Metric Summary Exception
Researchers	Analysis	Quality Growth Worksheet

Figure 3.3-3 Support to Personnel

The measurement concepts complement current Quality Assurance and testing practices. They are not a replacement for any current techniques utilized in normal quality assurance programs. For example, a major objective of quality assurance is to assure conformance with user/customer requirements. The software quality metric concepts described in this report provide a methodology for the user/customer to specify life-cycle-oriented quality requirement usually not considered, and a mechanism for measuring if those requirements have been attained. A function usually performed by quality assurance personnel is a review/audit of software products produced during a software development. The software metrics provide formality and quantification to these reviews/audits of the documents and code. The metric concepts also provide a vehicle for early involvement in the development since there are metrics which apply to the documents produced early in the development.

Testing is usually oriented toward correctness, reliability, and performance (efficiency). The metrics assist in the evaluation of other qualities such as maintainability, portability, and flexibility.

During the initial design phase of the AMT project, an informal requirements definition and operations concept reflecting the discussions in this paragraph were documented to ensure a common understanding between the RADC and USACSC project offices and the development team. These concepts were presented at a review at RADC. The informal requirements definition and operations concept were not called for in the statement of work, but were developed to supplement the requirements described in the statement of work. They were not formally delivered.

3.4 DESIGN OF AMT

The product of the design phase of the AMT project was a Design Plan, CDRL A001. The Design Plan contained a detailed design of the AMT, a Data Base Specification, a tool survey and a DBMS survey, the implementation schedule, and the plan for applying metrics to the AMT development. Each of these steps in the design of the AMT will be described in this section.

3.4.1 DESIGN GOALS OF AMT

There were specific design goals identified for the AMT prototype. A highly usable tool that could eventually operate in a number of environments in conjunction with other software and project management tools was desired. The software quality measurement framework was used to identify the design goals. Table 3.4.1-1 provides the formal definitions of the quality factors, from which three were chosen to be emphasized during the development of the AMT.

The statement of work identified Portability, Flexibility and Interoperability as important quality goals to be emphasized in the AMT development. They were identified as important because of the following facts:

(1) Portability was considered to be important because the AMT was developed for a Honeywell H6180/GCOS computer but may eventually be transported to Honeywell H6180/MULTICS, IBM 370/OS, and PDP 11/70/$^{\text{IAS}}$ environments.

(2) Flexibility was an important design consideration because the AMT is a prototype and additional reauirements will be forthcoming. Also the tool will eventually be used in a variety of environments, and have to process other languages besides COBOL.

(3) Interoperability was important because in any particular environment where the AMT might be used, other software tools might be available that are sources of metric information. We would want to be able to easily interface the AMT with these other tools to take advantage of the metric information the other tools collect automatically.

Table 3.4.1-1 Software Quality Factors

		FACTORS	DEFINITIONS
LIFE CYCLE STAGES	INITIAL PRODUCT OPERATION	CORRECTNESS	Extent to which a program statisfies its specifications and fulfills the user's mission objectives.
		RELIABILITY	Extent to which a program can be expected to peform its intended function with required precision.
		EFFICIENCY	The amount of computing resources and code required by a program to perform a function.
		INTEGRITY	Extent to which access to software or data by unauthorized persons can be controlled.
		USABILITY	Effort required to learn, operate, prepare input and interpret output of a program.
	PRODUCT REVISION	MAINTAINABILITY	Effort required to locate and fix an error in an operational program.
		TESTABILITY	Effort required to test a program to insure it performs its intended function.
		FLEXIBILITY	Effort required to modify an operational program.
	PRODUCT TRANSITION	PORTABILITY	Effort required to transfer a program from one hardware configuration and/or software system environment to another.
		RESUABILITY	Extent to which a program can be used in other applications - related to the packaging and scope of the functions that programs perform.
		INTEROPERABILITY	Effort required to couple one system with another.

3.4.2 DESIGN APPROACH

Specific design approaches were taken to satisfy these quality goals.

In part the modularity of the system design enhances the qualities of flexibility and portability. The AMT was divided into six functional subsystems: the Executive Services (EXS), the Data Base Management Services (DMS), the Automated Measurement Services (AMS), the Preprocessing Subsystem (PPS), the Report Generation Services (RGS), and the AMT Utility Services (UTL) as illustrated in the AMT hierarchy diagram shown in Figure 3.4-1.

The EXS provides the interface between the user and the AMT. The EXS interprets the user's commands and performs all of the necessary calls to the other AMT functions that actually carry out the actions requested by the user. To provide greater portability the AMT has its own Data Base Management Services (DMS). The DMS provides the capability to store and retrieve metric data from a random access file. The data base is described in more detail in Section 4 and in the Data Base Description Document, CDRL A011. The primitive operating system dependent functions of: opening a file, closing a file, reading a record from the file, and writing a record into the file are performed by the AMT Utility Services (UTL). These functions were isolated to facilitate modification for transporting the system to other environments. The Automated Measurements Services (AMS) extracts certain metric information directly from the COBOL source code. The Preprocessing Subsystem (PPS) is provided soley to support AMS functions. This subsystem has no direct interface to the user and is accessed only by the AMS. The basic function is a generalized parsing system to take source code input by an AMT user and generate parse trees representing that code.

The parse tree is then used by the AMS functions to produce values for the metrics worksheets. It should be noted that the preprocessing functions perform their own data management services, independent of the DMS functions. The reason for this separation is that the data that AMS and DMS functions individually manipulate are completely different in form and content and the AMS functions are the only functions that use or manipulate that particular type of data. The use of a generalized parsing function is a key design

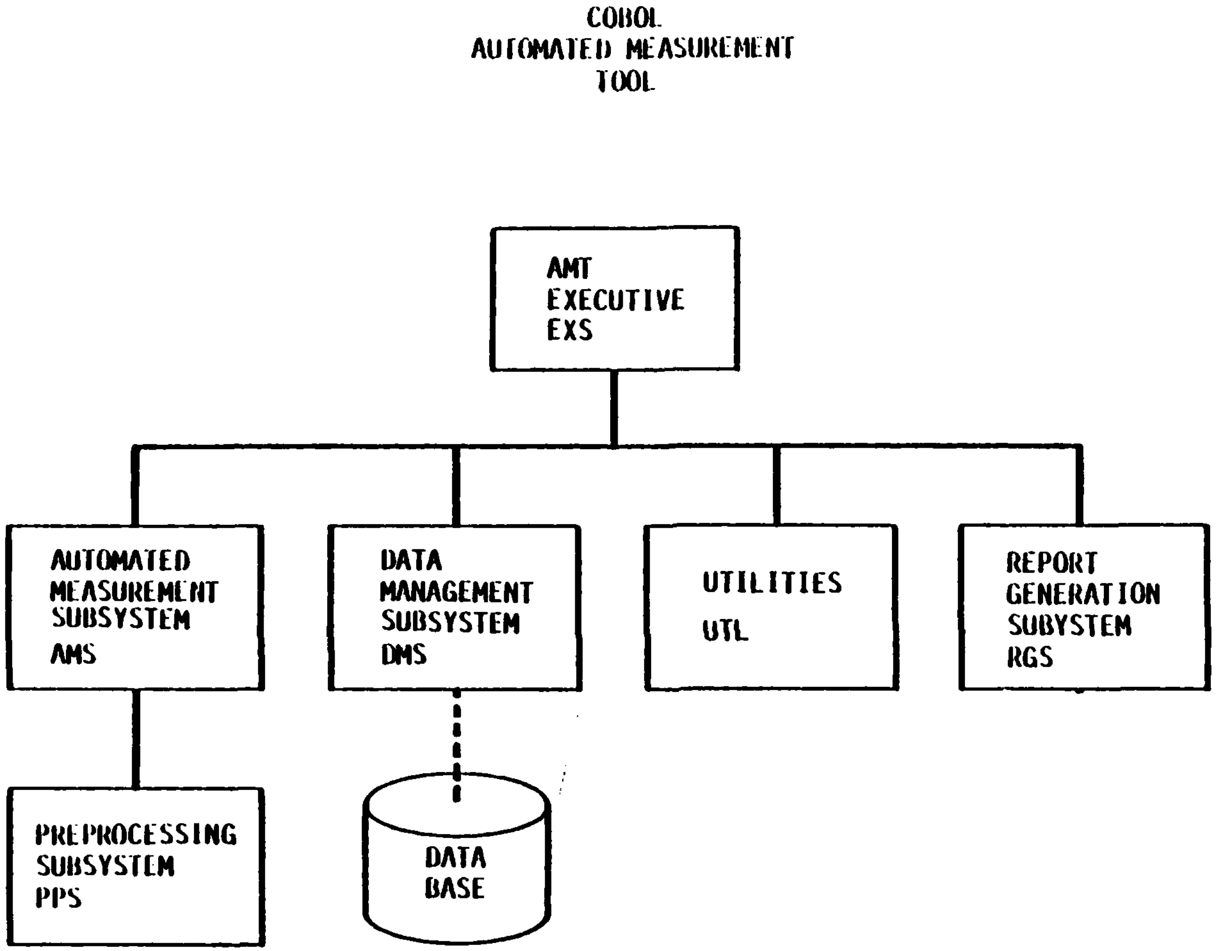


Figure 3.4-1 AMT Hierarchy Diagram

1075K-2

choice. By describing the grammar of a language other than COBOL and developing a scanner for that language, the parser will produce a parse tree representation that can be used by the remainder of the AMT functions with minor modification. Finally, the various metric reports and statistical analyses are performed by the Report Generation Services (RGS). We designed our own data management routines primarily to enhance the portability of the system. We conducted a Data Base management System survey on the target environments (HG180/GCOS, HG180/Multics, IBM 370/OS, PDP 11/70/IAS) to evaluate the portability issues. Certainly if one DBMS had existed on all four environments our portability problems with respect to data would have been solved. However, this was not the case and, in fact, our analysis determined that developing our own data management routines would be much less expensive than having to convert from one DBMS to another when we wanted to move the AMT to another environment. The goals of the Data Management Services were:

- The data base must be portable not only in the transfer of the data base functions from one machine to another, but the data within the data base needs to be portable for research reasons.
- The data must be easy to access and insert.
- The user must be able to enter and exit the data base with ease.
- The data base needs to be highly maintainable.
- The data base needs to be accessible by other software tools in a given software development environment.
- No on-line access via a general query language was provided.

Appendix C contains the results of the DBMS survey.

Interoperability was designed into the system primarily through the data base design. By providing routines to manipulate the data with respect to the AMT data base, the output of other software tools could be accessed and inserted into the AMT data base. Thus if the AMT was being used in an environment where PSL/PSA was utilized, the output of PSA could be searched by a software routine and appropriate data extracted and inserted into the AMT data base. We conducted a tool survey to identify potential tools for interfacing with the AMT. Several potential tools were identified. The results are in Appendix D.

During the design phase, an implementation schedule was developed. The approach taken will be described in the next paragraph. Also, during the design phase, we began applying metrics to the development effort. This application was an experiment to assess the effectiveness of applying metrics during a development. The approach and results of the experiment are in Section 5 of this report.

The actual design of the AMT was conducted using structured design techniques. The design of the system was iteratively decomposed to more detailed descriptions of the subsystems shown in Figure 3.4-1 and eventually to detailed designs of each module. Hierarchy charts for each subsystem were generated, HIPO diagrams were constructed for each module, and program design language descriptions (PDL) of the logic were prepared. The PDL used was an Ada-like language developed by General Electric. Complexity measures were automatically calculated from the PDL's. These metrics were utilized during design team walkthroughs to evaluate the overall complexity of the design. The design was documented in the Design Plan (CDRL A001) and in part in the System/Subsystem Description Document (CDRL A009).

3.4.3 USER ORIENTED CONCEPT

Experience with software quality metrics has pointed out that a variety of personnel have use of the metric data for a variety of reasons. Program Managers are interested in the progress of a project with regard to quality goals. Developers use metric data to aid design decisions and enforce standards. Test personnel use the data to determine test strategies, emphasis, and level of effort.

The quality assurance staff monitors and reports on violations to standards, goal achievement, and quality compliance. Researchers are typically interested in historical data.

Given these variety of users, ranging from sophisticated to uninitiated, from experienced to untrained, the tool was designed with the following charateristics in mind:

- Transparent: The actual operations of the tool should not interfere with the use of the tool.
- System independent: The user should have to use a minimum of system language to use the tool.
- Forgiving: The tool should have the capability of trapping errors and recovering gracefully to minimize the impact on the user.
- Helpful: The software should carry as much interactive training functions as possible within the tool.
- User flexible: For the experienced user of the tool, options should be available to increase the speed with which the user can interact with the system.
- In general the tool should be "user friendly."

A User's Manual (CDRL A012) was developed to provide guidance to users on the use of the AMT.

3.5 IMPLEMENTATION APPROACH

The implementation of the AMT was done in IFTRAN2, a General Research Corporation structured programming preprocessor to FORTRAN. IFTRAN2 provided a structured FORTRAN-based language for developing the source code. As a result, the code is well structured and easier to read. The implementation was conducted incrementally over an 11 month period. The initial capability developed was the Executive Services Subsystem. This subsystem processes the user command language. By providing this function first, future users could begin training and gain familiarity with the user interface. Also the Preprocessing Subsystem was started early during the implementation phase. The parser portion of this subsystem was an existing system. We had to describe the COBOL grammar in a Backus-Naur-Form (BNF) like language, however, and because we could find no such description of COBOL we started this task early to avoid unexpected difficulties.

The next subsystem to be developed was the Data Management Services. This allowed demonstration of the capability to manually enter and extract data from the data base. The Utilities Subsystem was also developed at this time .

The Automated Measurement Subsystem and Report Generation Subsystem were then completed to provide the full operational capability.

Documentation of the implementation of the AMT was provided in the Program Specification (CDRL A010), the Program Maintenance Manual (CDRL A013), and the source code provided.

Because of the inefficiencies of developing the software remotely on the RADC H6180, the AMT subsystems were prototyped on a General Electric VAX 11/780. To enhance the portability of these prototypes to the RADC H6180, specific standards and conventions were developed and followed. These standards and conventions are described in Appendix B. Final system integration and modification was done remotely on the RADC H6180.

3.6 AMT TEST

A Test Plan (CDRL A015) was developed and submitted for review by RADC and USACSC at the end of the design phase. This test plan incorporated a strategy of testing each subsystem increment as it was developed, integrating them stepwise into the development environment, performing system test, transferring to the RADC target environment, and performing regression system tests. The RADC testing was done remotely. The tests were based primarily on demonstrating the functional capabilities of the AMT and its error handling capabilities, using five COBOL programs provided by the USACSC. The results of the tests were documented in a Test Analysis Document (CDRL A014)

3.7 AMT TRAINING

The last phase of the project was to develop a training outline, provide training in the form of a demonstration at RADC, and deliver the tool and documentation, including the analyses performed. The training material (CDRL A006) was delivered also. The AMT User's Manual (CDRL A012) provides examples from operating the AMT.

SECTION 4
AMT DESCRIPTION

4.1 OVERVIEW

The AMT is a system to be used in support of the quality assurance function during software development and maintenance. Four types of users are envisioned for the system, a customer or user of a software system to be developed, a quality assurance analyst, the software development project manager, and a software researcher. The objective of the system is to provide quality assurance information to these users in the form of software quality metrics.

The system provides five major services to accomplish its function of providing software quality metrics to its users.

- Automatic Metric Collection
- Manual Metric Collection
- Storage and Retrieval
- Report Generation
- User Messages

In order to support these services, the subsystem design shown in the functional block diagram in Figure 3.4.1-1 was developed. The major functions are:

- Executive Services (EXS)
- Database Management Services (DMS)
- Automatic Measurement Services (AMS)
- Report Generation Services (RGS)

The relationship between the services provided by the AMT and the functional areas is shown in Figure 4.1-1. The following paragraphs describe each of the functional areas. Detailed hierarchy charts of each subsystem are contained in the System/Subsystem Specification (CDRL A009).

FUNCTIONAL AREAS

User Services	EXS	DMS	AMS	RGS
Auto. Metric Collection		X	X	
Manual Metric Collection	X	X		
Storage and Retrieval	X	X		
Reports	X	X		X
User Messages	X			

Figure 4.1-1 Services Provided By Functional Areas

4.2 EXECUTIVE SERVICES

The function of the Executive Services (EXS) is to provide the interface between the user and the AMT. Thus, EXS interprets the user commands and performs all the necessary calls to other AMT functions to actually carry out the actions indicated by the user with his command. The EXS monitors system status and reports all errors trapped by itself or any other subsystem. Built in debugging functions which allow a user/programmer to trace real-time program execution are also controlled by the EXS. The EXS queries for complete command information and also provides user 'help' information.

4.2.1 COMMAND LANGUAGE

The command language processed by EXS is as follows:

CR Carriage Return
Responds with prompt.

CREATE databasename
Creates a file for storing worksheet data. Uses system file manipulation routines.

DELETE Modulename
This command deletes a module from the current database

E Exits the user from the current task.

END Terminates AMT session. Closes all open files prior to termination.

ENTER modulename
Used to identify new modules for which data is to be stored in current data base.

GET worksheet number [sectionnumber] [itemnumber] [modulename]
Retrieves items currently stored in data base. Retrieval is based on individual items, sections from a worksheet, or an entire worksheet. For worksheets that are at module level, module name must be specified.

HELP [commandname]
Provides text which explains syntax of each command and function. Without command name, it provides list of available commands.

MEASURE sourcefilename modulename
Causes automated source code metric collection to be initiated using the source code in sourcefilenamefile. Data collected is stored in appropriate worksheet for module name.

PUT worksheetnumber [sectionnumber] [itemnumber] [modulename]
Allows storage of values in the database. The system prompts operator for value or values by placing worksheet identifying phrase or question on terminal. Prompts are for individual items, or for each item within a whole worksheet. For worksheets that are organized by module, the modulename must be entered or if it is not entered, a prompt requesting it is displayed.

REPORT reportname [Printer]
Generates the report requested. The reports are presented at the terminal. Certain reports require further input and the operator is prompted for further input.

SET databasename
The data base for subsequent commands to interact with is identified by this command. Only one database may be processed at any one time. A SET command supercedes previous SET'S. The data base had to have been created for the SET to work.

[ ] indicates optional data

4.2.2 GENERAL CONVENTIONS FOR ENTERING A COMMAND

When entering commands and keywords the user need only type the first three characters of the command name, e.g., CRE for CREATE. the one exception to the rule is the worksheet numbers, for which four characters are required i.e., WS2A and WS2B for worksheet 2A or worksheet 2B.

The various parts of a command must be separated by one or more spaces i.e., CRE PROG1. For commands that have parameters the user may type just the command name followed by a carriage return. The AMT will prompt the user for the necessasy parameters. For example, when CRE is entered AMT prompts with ENTER DATABASE NAME: Each command or command part must be terminated with a carriage return.

When entering responses to the Yes/No questions a "Y" may be typed for "Yes", and "N" may be typed for a "No". Not applicable responses must be indicated by entering "NA".

4.3 DATA BASE MANAGEMENT SERVICES

The Database Management Services (DMS) provide for the storage and retrieval of raw metric data. The ability to create, open, close, and expand AMT data bases is also provided by this subsystem. While individual data base items can be referenced, the data structure used most often by DMS is the worksheet. This implementation provides rapid and efficient access to the data base entries and their values. System dependent functions such as file handling and disk access are isolated in the Utilities Subsystem (UTL).

4.3.1 DATABASE DESIGN

The worksheet oriented organization of the metrics. The worksheets are defined in the Software Measurement Manual [MCCJ19]. Samples are in Appendix A.

The logical structure of the data base is described in Figure 4.3-1. A separate data base is maintained for each system entered by a user or users of the AMT. Logical records correspond to worksheets, with two distinguished worksheets (1 and 2a) for which only one copy each exits. This is because these worksheets are system level and refer to all modules. Multiple copies of worksheets 2b and 3 are provided since these correspond to module level metrics. Reference is made by worksheet number, section number and item number and, in the case of worksheets 2b and 3, by module name.

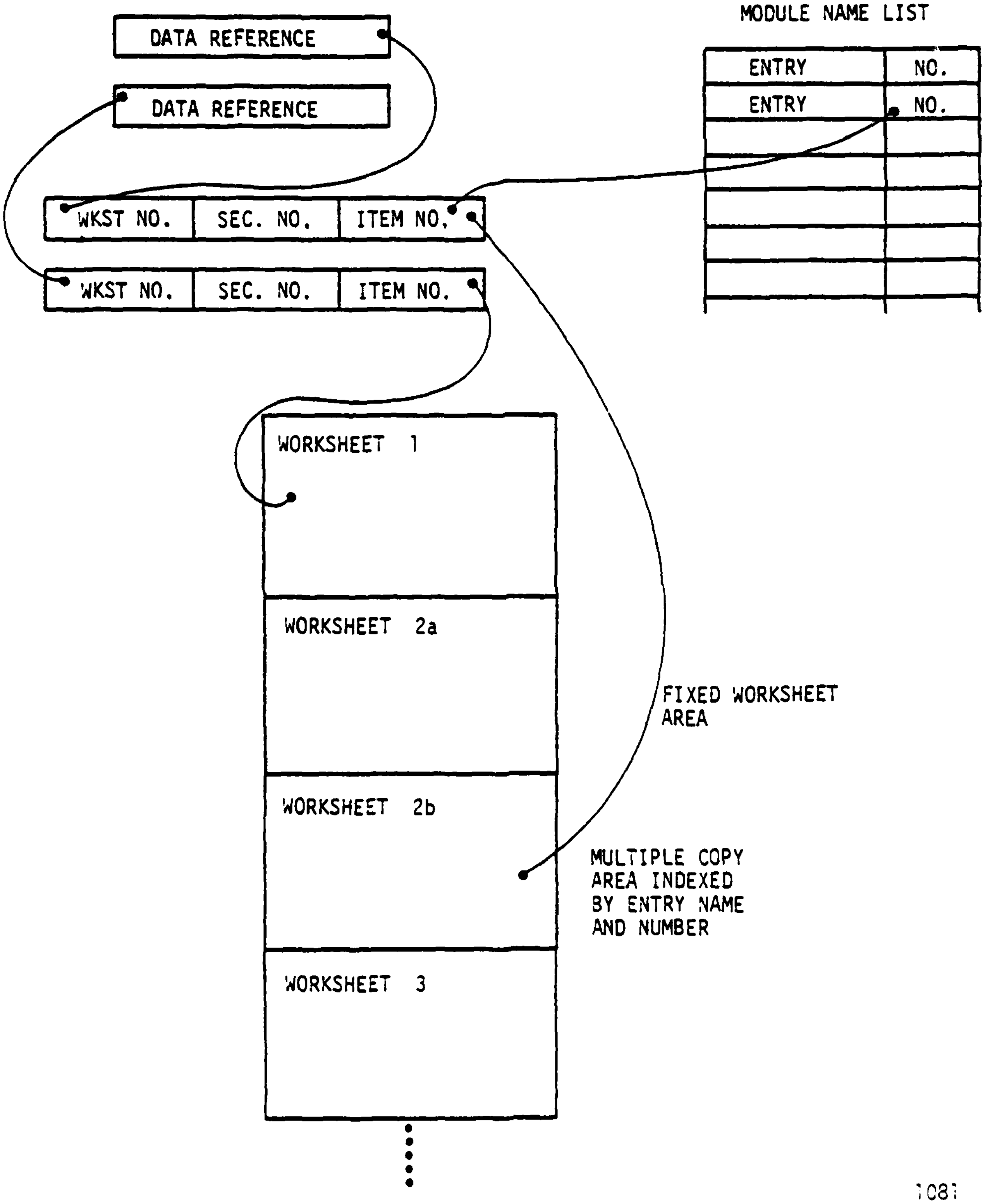
Individual worksheet data items occur as elements of an array associated with each worksheet/logical record. Logical records are linked by number to a particular module name, the module names are kept in a list which is physically stored at the beginning of the data base file. The prototype version of the AMT has a limitation per data base of 50 modules. The system organization of the data base using the GCOS file management system is shown in Figure 4.3-2.

The logical record number of a particular module's worksheet is thus calculated by finding the corresponding entry number of the module in the module name list and adding an offset number (which is implementation dependent). Particular data items can then be obtained by using the array index associated with the item. Array indices for a particular data item in a worksheet are stored in a pointer table which is indexed by the triple (worksheet number, section number, item number). This table is pre-defined and is stored as a DATA statement (see Figure 4.3-3). A more complete description of the data base implementation is in the AMT Data Base Description document (CDRL A011).

Raw metric data is stored in the data base in two ways. First, it can be stored manually by the user. The user can enter module names using the ENTER command. This command enters the module name in the data base and reserves a designated area for storing worksheets 2b and 3's data for that module. The user can also enter metric raw data from one of the worksheets by using the PUT command. The PUT command can be used to prompt the user for one of the data items in a section or a worksheet or it can be used to enter one specific value individually.

These metric values are calculated each time a user tries to generate reports. This recalculation of the metric values is performed to insure current values. The slight processing overhead is considered worth the benefit of currency. The metric values calculated are stored in local arrays. The system level metrics are in single dimension arrays while the module level metrics are in two dimension arrays. This physical and logical structure allows for the processing flow within AMT to be as shown in Figure 4.3-4. The reason raw metric data is maintained instead of just the calculated metric values is to allow researchers to change the calculations of metrics to investigate other algorithms.

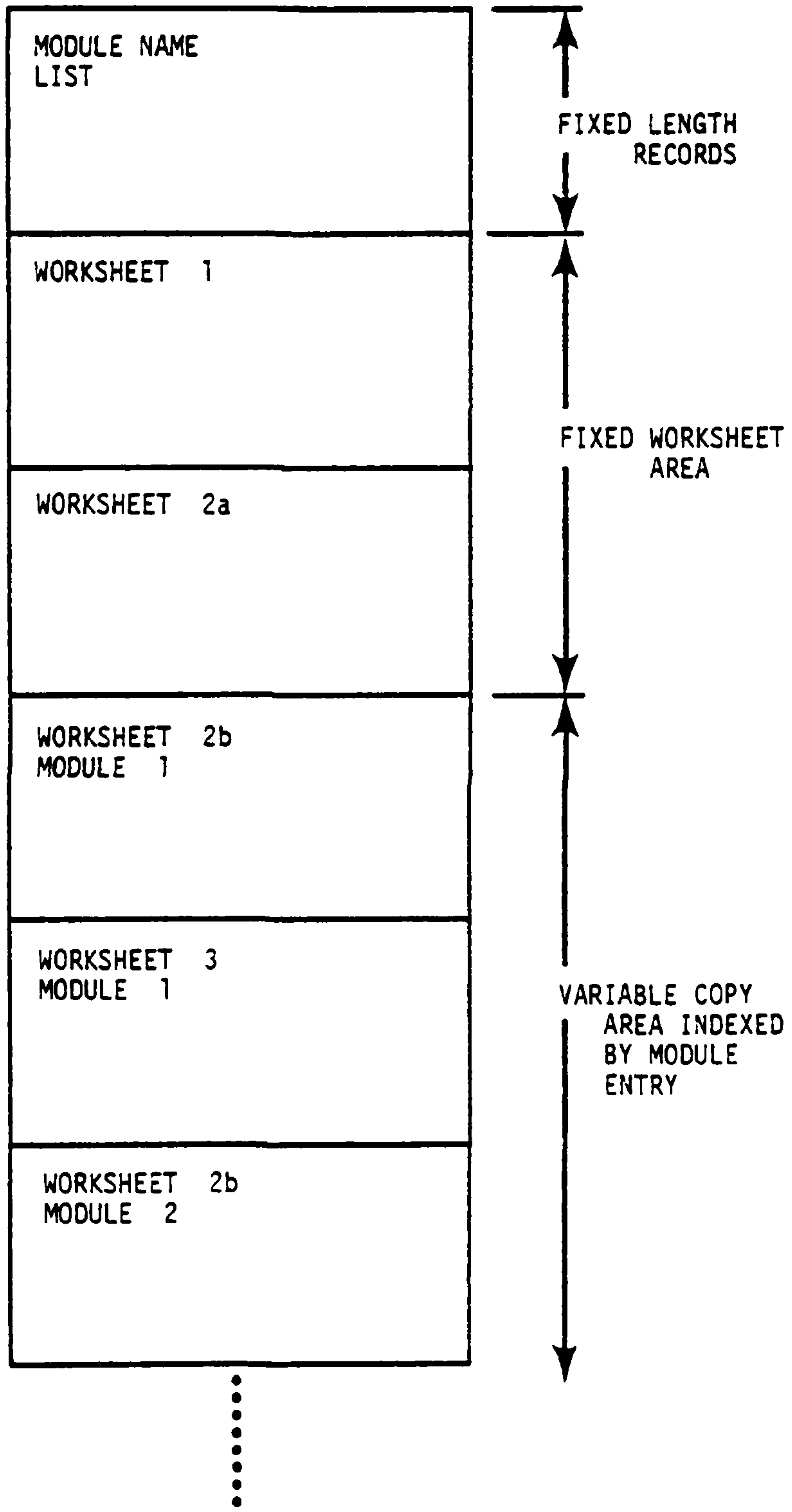
DATA MANAGEMENT SERVICES (DMS)
LOGICAL DESCRIPTION OF DATABASE



1081

Figure 4.3-1 Logical Description of AMT Data Base

PHYSICAL ORGANIZATION OF DATABASE



1079

Figure 4.3-2 Physical Organization of Data Base

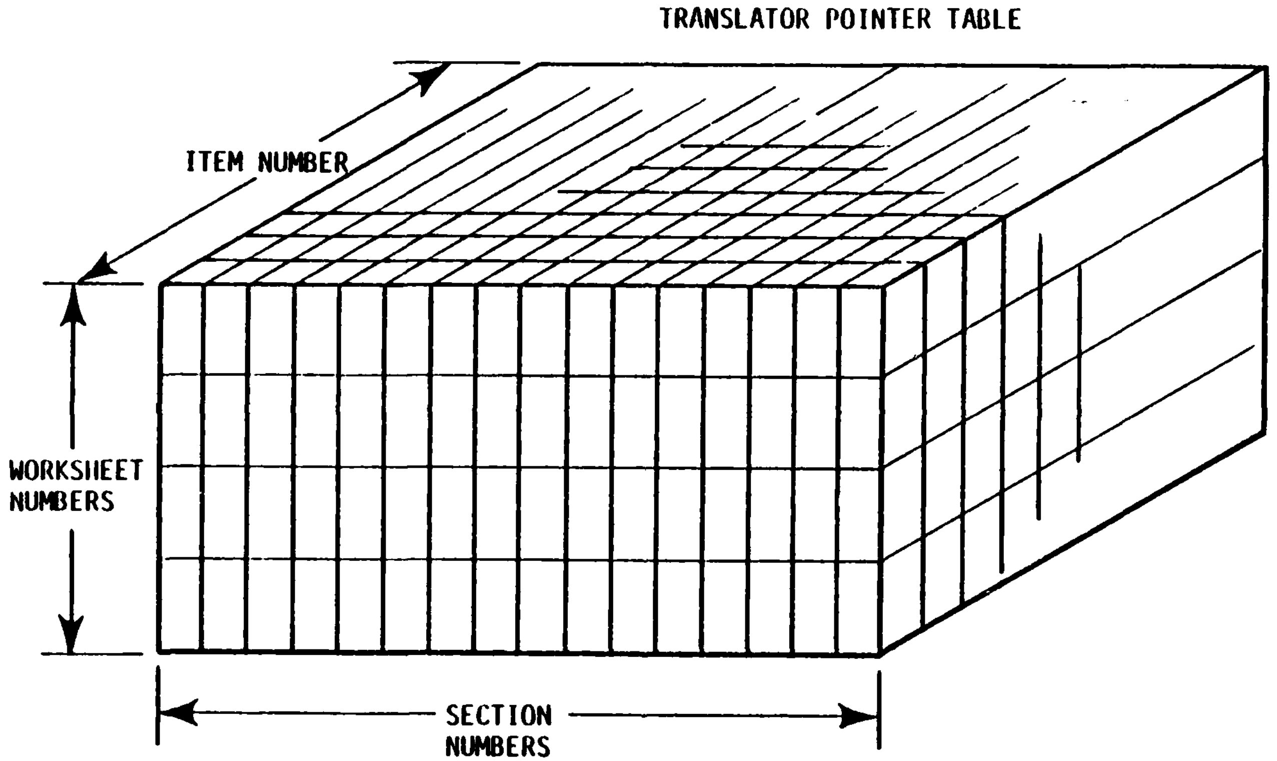


TABLE (WS, Sn, in) = ARRAY INDEX

Figure 4.3-3 Pointer Table

1077

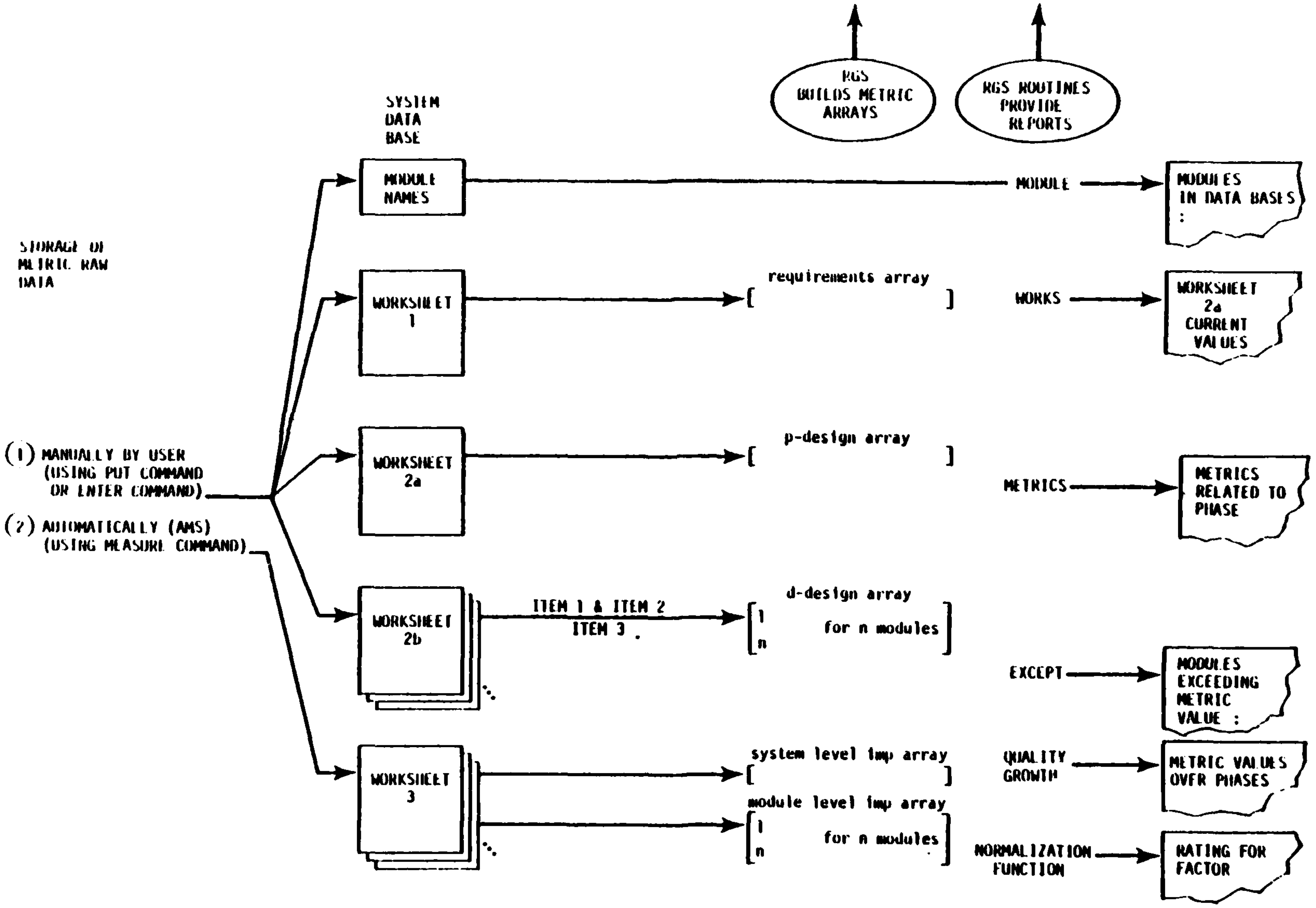


Figure 4.3-4 AMT Processing

4.3.2 FILE SPECIFICATION CONVENTIONS

To utilize a data base of metric information within the AMT, the CREATE and SET commands are used. When using these commands the user is interfacing with the AMT data base. The AMT takes the character string the user enters as part of the CREATE of SET command and appends .DAT to it. For example:

```
        CRE TEST
```

creates a data base file called TEST.DAT. At other times, the user may want to interface with a file other than an AMT data base. An example is the MEASURE command. In this situation the AMT accesses a file which has source code that is to be analyzed by the AMT parser and Automated Measurement Subsystem. That file was established using the system editor and file management system. In this case, the user must specify the full filename of the file containing the source code.

For example:

```
        MEASURE PROG1.CBL MOD1
```

accesses a file called PROG1.CBL which contains the COBOL source code for MOD1. The user should be aware of the file specification/naming conventions and file maintenance procedures of the computer they are using to run AMT in order to name and maintain the AMT files.

4.4 AUTOMATED MEASUREMENT SERVICES

The amount of data that can be automatically collected is limited to data that can be derived from machine readable sources such as design materials generated on the computer and the actual source code. This paragraph describes the current automatic data collection capability of the AMT.

The current version of the AMT automatically collects and stores raw data from COBOL source code. The remainder of the raw data required to calculate the metrics must be manually collected.

4.4.1 AUTOMATED MEASUREMENT OF SOURCE CODE

Currently the AMT collects data from COBOL source code for individual modules. A total of 25 different measurements are collected automatically. These measurements can be divided into the following broad classes:

1. Counts of total number of code statements and comment statements;
2. Counts for individual types of statements e.g., input, output, exit, entry, declarative, etc,;
3. Counts of different types of branching statements both conditional and unconditional; and
4. Counts of operands and operators, for use in calculating Halstead's measure.

The specific data items automatically collected are shown in Table 4.4.1-1. Also noted in the table is the entry on the worksheet that this item relates to and the metric that it helps calculate. The worksheet entry is identified by worksheet number (WS3), section number (SI), and item number (I1). Thus a notation such as WS3, SI, I1 is read as worksheet 3, section I, item 1.

This automated support accounts for 25 of the 92 worksheet 3 data items, or 27% of the measurements required at the implementation phase of a development. These 25 data items help calculate 9 of the 38 metrics related to implementation, or 24% of those metrics. The metric calculation is described in paragraph 4.5, Report Generation.

The automated data collection is performed by the Automated Measurement Services (AMS) and the Preprocessing Services (PPS) subsystems. The user invokes these subsystems using the MEASURE command. The Preprocessing Services uses an LL(1) generalized parser to decompose the COBOL source code contained in the file identified by the MEASURE command. The result of the parsing is a parse tree representation of the source code. A description of the parser, which uses a Backus-Naur-Form description of COBOL grammar, is in the AMT System/Subsystem Specification document (CDRL A009).

The Automated Measurement Services subsystem traverses the parse tree and counts the various data items and enters them in the data base. More detailed descriptions of the design of these subsystems are in the Design Plan, CDRL A001.

Table 4.4.1-1 Automated Metric Data

	Worksheet	
Data Item	Entry	Metric
1. Number of lines of code	WS3,SI,I1	MO.2 Modular Implementation
2. Number of lines of code minus comment statements	WS3,SI,I2	SI.4 Coding Simplicity
3. Number of declarative statements	WS3,SI,I4	SI.4 Coding Simplicity
4. Number of data manipulation statements	WS3,SI,I5	SI.4 Coding Simplicity
5. Number of statement labels	WS3,SI,I6	SI.4 Coding Simplicity
6. Number of entrances to modules	WS3,SI,I7	SI.1 Design Structure
7. Number of exits from modules	WS3,SI,I8	SI.1 Design Structure
8. Maximum nesting level	WS3,SI,I9	SI.4 Coding Simplicity
9. Number of decision points	WS3,SI,I10	SI.3 Complexity
10. Number of subdecision points	WS3,SI,I11	SI.3 Complexity
11. Number of conditional branches	WS3,SI,I12	SI.4 Coding Simplicity
12. Number of unconditional branches	WS3,SI,I13	SI.4 Coding Simplicity
13. Number of loops	WS3,SI,I14	SI.4 Coding Simplicity
14. Number of module modifying statements	WS3,SI,I17	SI.4 Coding Simplicity
15. Number of operators	WS3,SII,I1	CO.1 Conciseness
16. Number of operands	WS3,SII,I2	CO 1 Conciseness
17. Number of unique operators	WS3,SII,I3	CO.1 Conciseness
18. Number of unique operands	WS3,SII,I4	CO.1 Conciseness
19. Number of comments	WS3,SIII,I1	SD.1 Quantity of Comments
20. Number of continuation lines	WS3,SIII,I13	SD.3 Language Descriptiveness
21. Number of input statements	WS3,SIV,I1	MI.1 Machine Independence
22. Number of output statements	WS3,SIV,I2	MI.1 Machine Independence
23. Number of calls to modules	WS3,SV,I1	MO.2 Modular Implementation
24. Number of mixed mode expressions	WS3,SVIII,I1	EE.3 Executive Efficiency
25. Number of variables initialized when declared	WS3,SVIII,I2	EE.3 Executive Efficiency

The significant aspect of this approach is that other language grammers can be described to the parser, a scanner developed, and the parser will produce a parse tree representation of the other language. With careful attention paid to the token representation, the AMS will be able to process this parse tree representation of the other language with little or no modification.

4.4.2 AUTOMATED AID TO MANUAL MEASUREMENT OF METRICS

Paragraph 4.4.1 described the 25 measurements that are automatically collected by the AMT. There are 31 measurements during requirements, 173 during design, and 67 during implementation that are not currently automatically collected. The AMT does provide some automated support to their collection and storage in a data base.

The AMT will automatically generate worksheet forms which must be filled out by an analyst/inspector. These worksheets can be printed with any current data that exists in the data base displayed. This is particularly useful for worksheet 3 which will be partially completed automatically by the AMT when the MEASURE command is used.

The AMT also provides the PUT command that facilitates the user entering data into the data base. The PUT command is described in paragraph 4.3 and in the AMT Users Manual (CDRL A012).

The standard procedure for using the AMT to assist in manual collection of metric data is as follows:

(1) The appropriate worksheet will be printed at the terminal. This worksheet will be the data collection form for the inspector's use.

(2) Reference should be made to the Metrics Enhancement Final Report [MCCJ79VolI], Software Quality Measurement Manual [MCCJ79VolII], and the AMT User's Manual [CDRL A012]. These references provide a description of the metrics, the worksheets, and how they can be used in context of the AMT respectively. The AMT User's Manual provides a copy of each worksheet (Appendix C), instructions for completing the worksheets (Appendix D), and an example of the worksheets completed for a COBOL source program.

(3) The appropriate source material should be gathered. For example, the Requirements Specification is the source material for worksheet 1.

(4) The source material should be briefly read for both format and content.

(5) A detailed analysis of the source material should then be conducted using the questions on the worksheet as directed inspection of the source material.

(6) Once all questions are answered, then the inspector should document any overall observations that might be made based on the inspection.

(7) The answers to the worksheet questions should then be entered into the AMT data base using the PUT command

This standard procedure will become routine with application experience. This is especially true if the experience is gained in an environment where the documents prepared follow a consistent format or are prepared according to the same military standard. In these cases, the information sought as a result of the worksheet questions typically will be in a certain section of the document.

This general approach provides the inspector a framework in which to inspect material, specific questions to answer, and a directed sequence to follow. This consistency and quantification in the inspection process enhances the consistency between inspectors and makes the process more repeatable and consistent between applications. These benefits provide better inspection results, more feedback to the developers and management, and therefore aid in achieving a higher quality software product.

4.4.3 USE OF OTHER AUTOMATED TOOLS

The AMT was developed with the concept of eventually interfacing it to other software tools in a software development environment. The interfacing would be done by extracting metric data available from the processing done by the other tools and inserting the data into the AMT data base so metrics could be calculated.

A program would have to be written which extracts the appropriate data from the output file of a tool and using the AMT PUT command, inserts it into the data base. Potential tools that should be considered are Requirements

Specification Language processors/analyzers, Program Design Language Processors/analyzers, code auditors, code instrumentors, and configuration management tools.

4.5 REPORT GENERATION SERVICES

The Report Generation Services (RGS) provides the user the ability to generate various reports that reflect the contents of a database. Nine reports may be requested by the user to display the metric data in a variety of formats, and by performing additional calculations, present various forms of data both at the module and system levels. The processing that is done is shown in Figure 4.3-4. Basically, data is extracted from the data base to calculate metric values. The algorithms for performing these calculations are contained in the Report Generation Services routines. These algorithms are defined in the Metrics Enhancement Final Report [MCCJ79] and in the Program Specification Document (CDRL A010). Samples of these reports are included in the Users Manual and in Appendix A. Brief descriptions follow:

4.5.1 MODULE REPORT

This report displays the catalog of modules that have been entered into the database. It provides a status report on the database.

4.5.2 METRIC REPORT

This report calculates the value of each metric catagorized by factor and by development phase. This report is used to determine a total picture of the project as measurements are taken.

4.5.3 EXCEPTION REPORT

The exception report delivers the relationship of each module to a given threshold value of a particular metric. The relationship (less than, equal to, or greater than) and the threshold value is input from the user. This report can be used to identify modules whose scores do not meet a certain threshold, identifying them as potential problems.

4.5.4 QUALITY GROWTH REPORT

When the user wishes to track the value of a particular metric over time, the Quality Growth Report will furnish a tabular display of the scores of a selected metric over the phases of the project. This report is used to track a particular metric through a project to see how its value changes.

4.5.5 NORMALIZATION REPORT

The Normalization Report provides the user with the overall rating of a selected quality factor. A series of regression equations are displayed which have been empirically derived from research. The current metric values are substituted in the equations and a rating for the selected quality factor is calculated. Regression equations exist for the quality factors Reliability, Maintainability, Portability, and Flexibility only. The normalization function is calculated at a module level.

4.5.6 STATISTICS REPORT

The Statistics Report provides a profile of COBOL constructs for each module.

4.5.7 SUMMARY REPORT

The summary report provides a summary of the metric scores for all of the modules in the system.

4.5.8 WORKSHEET REPORT

The worksheet report displays the raw data entered in each worksheet. It represents the current values in the database. It is used to verify and track data entry.

4.5.9 MATRIX REPORT

This report displays the average and standard deviations for all metrics values for all modules. This report displays all of this information in a matrix form allowing the user to easily identify modules with metric scores that vary from the system average.

4.5.10 REPORT SUMMARY

The reports may be classified as to their primary use:

- Descriptive
- Historical
- Diagnostic

The reports that are descriptive are the Summary, Matrix, Module, and Metric reports. Their common characteristic is that they report data in a format implying no judgements concerning the data. The Summary Report reports all metric scores for each module or for all the modules. The Matrix Report displays the mean and standard deviation of the modules for each metric. It is a good snapshot of the data in the data base. The Module Report is meant for operational personnel. It reports those names of the modules which are in the data base. The Metric Report is a more detailed output which displays the metric values for each module in a detailed form.

The Historical Reports are the Quality Growth and Worksheet Reports. The Quality Growth report provides the quality trend of a module through the development phases. The Worksheet Report gives a very detailed display of the raw data before it is transformed into metric scores. It's main use is to track data entry and updates.

The Diagnostic Reports are those that identify potential problem areas. They are the Normalization, Exception, and Statistics Reports. The Normalization Report applies the regression equations derived from research to metric values related to the quality factors of flexibility, maintainability, portability, and reliability at the module level. These regression equations have been developed through examination of previous projects.

Regression equations for the remaining quality factors have not been established. The Exception Report provides a comparison of the metric scores with predetermined, user supplied values. The Statistics Report gives a diagnostic snapshot of any module. These data may be used to evaluate standards or identify potential problem areas.

The typical use of these reports is described in Table 3.3-3. The table identifies what type of support each report offers different job functions.

Additional reports can be added with relatively minor effort. Reference to the report would have to be added into the Executive Services processing and a report routine written using the GET command to extract appropriate data from the data base. Reference should be made to the AMT Maintenance Manual (CDRL A013) and the AMT Program Specification document (CDRL A010) for further insight into the modifications necessary to write a new report.

SECTION 5
RESULTS OF QUALITY METRIC EXPERIMENT

5.1 INTRODUCTION

The software quality metrics concepts were applied during the development of the Automated Measurement Tool. This is the first formal application of the software metrics defined in [MCCJ77A] and therefore viewed as an experimental demonstration of the metric concepts. The purposes of this application of the metrics were:

(1) Provide additional experience with applying the metrics and validating their utility. An added benefit of this experience is that the application of the metrics was in-line with the development effort and not after the fact as past applications have been.

(2) Provide quality assurance feedback to the development team and the RADC project engineer. The application of the metrics was planned as a complement to the planned testing to insure production of an effective software product.

(3) Meet the quality requirements identified in the statement of work. Some specific qualities were identified as being important to the AMT development and the metrics were applied to provide some assurance that emphasis was placed on their inclusion.

(4) Provide an experimental basis for generating suggestions on how to best use the metrics in a contractual environment.

This section describes the approach to performing this experimental application of the concepts, the results achieved, and lessons learned.

This section has the following organization. Paragraph 5.2 describes the quality goals identified for the contract. Paragraph 5.3 describes the process followed in applying the worksheets during the development. Paragraph 5.4 describes the results of the application of the metrics in terms of scores achieved, observations made based on the scores, assessment of the metrics, calculations of the normalization functions, and comparison of the scores with the goals established. Paragraph 5.5 compares the metric values achieved during the AMT development with those observed in past experiences. Paragraph 5.6 summarizes the lessons learned from the experiment.

5.2 QUALITY GOALS FOR THE AMT DEVELOPMENT

5.2.1 STATEMENT OF WORK RELATED QUALITY GOALS

A high level statement of the quality goals of the AMT development was mentioned in paragraph 4.1.1.3 of the statement of work.

The quality factors portability, flexibility, and interoperability were identified as important to the AMT product. Portability was important because the system design must consider four different target environments: H6000/GCOS, H6000/MULTICS, PDP 11/70 UNIX, and IBM 370/OS.

Flexibility was important because a subset of the entire set of metrics will initially be automatically collected. The system may be extended in the future to collect a larger subset. Interoperability was important because a number of new or existing software tools may be interfaced with the system.

5.2.2 SPECIFIC QUALITY GOALS ESTABLISHED

To establish more specific goals against which to measure the development effort, the Software Quality Requirements Survey Form, shown in Table 5.2.2-1, was completed by a representation of RADC, USACSC/AIRMICS, and the development team. A form of the Delphi technique was used in that each individual completed the form and then in a group session agreed to a priorized list of the quality factors important to the AMT development. The quality goals decided upon are identified in Table 5.2.2-2.

The first six factors: Portability, Flexibility, Reusability, Interoperability, Correctness, and Maintainability were especially emphasized in the experiment since they were ranked highest.

Additionally, specific ratings for four factors and for several metrics were established. The ratings, shown in Table 5.2.2-3, were established based on previous experience and are set at the specific levels indicated as part of the experiment. The previous experience refers to the validation efforts conducted under preciously funded research efforts. The Software Quality Measurement Manual [MCCJ79] provides additional guidance on certain metrics and normalization function thresholds.

Table 5.2.2-1 Software Quality Requirements Survey Form

1. The 11 quality factors listed below have been isolated from the current literature. They are not meant to be exhaustive, but to reflect what is currently thought to be important. Please indicate whether you consider each factor to be Very Important (VI), Important (I), Somewhat Important (SI), or Not Important (NI) as design goals in the system you are currently working on.

RESPONSE	FACTORS	DEFINITION
________	CORRECTNESS	Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
________	RELIABILITY	Extent to which a program can be expected to perform its intended function with required precision.
________	EFFICIENCY	The amount of computing resources and code required by a program to perform a function.
________	INTEGRITY	Extent to which access to software or data by unauthorized persons can be controlled.
________	USABILITY	Effort required to learn, operate, prepare input, and interpret output of a program.
________	MAINTAINABILITY	Effort required to locate and fix an error in an operational program.
________	TESTABILITY	Effort required to test a program to insure it performs its intended function.
________	FLEXIBILITY	Effort required to modify an operational program.
________	PORTABILITY	Effort required to transfer a program from one hardware configuration and/or software system environment to another.
________	REUSABILITY	Extent to which a program can be used in other applications - related to the packaging and scope of the functions that programs perform.
________	INTEROPERABILITY	Effort required to couple one system with another.

2. What type(s) of application are you currently involved in?

____

3. Are you currently in:

____________ 1. Development phase
____________ 2. Operations/Maintenance phase

4. Please indicate the title which most closely describes your position:

____________ 1. Program Manager
____________ 2. Technical Consultant
____________ 3. Systems Analyst
____________ 4. Other (please specify)____________

Table 5.2.2-2 Quality Requirements for AMT (In Order of Ranking)

FACTOR		CONSIDERATION
PORTABILITY	(VI)	Targeted for IBM 370, H6000, PDP 11/70.
FLEXIBILITY	(VI)	Tools to be added and capabilities enhance.
REUSABILITY	(VI)	In transporting to other environments and languages, want to reuse as much software as possible.
INTEROPERABILITY	(I)	Software tools to be interfaced with.
CORRECTNESS	(I)	Utility of AMT depends on its functioning correctly.
MAINTAINABILITY	(I)	May eventually be maintained by personnel other than developers.
RELIABILITY	(SI)	Accuracy of metric counts and quality rating calculations important.
USABILITY	(SI)	To be used by managers and QA analysts.
TESTABILITY	(SI)	The correctness of the metric data collection must be demonstrated.
INTEGRITY	(NI)	The security of *the data base is not really* critical.
EFFICIENCY	(NI)	Processing efficiency not critical.

Where VI is Very Important
I is Important
SI is Somewhat Important
NI is not important

Table 5.2.2-3 Specific Quality Goals

FACTOR RATINGS	AMT	C^2	MIS
Flexibility	.7	.4	.4
Portability	.75	NM	.6
Maintainability	.7	.33	.20
Reliability	.9	.98	.92

METRIC THRESHOLD VALUES		
MO.2	Modular Implementation Measure	.7
GE.2	Generality Checklist	.35
SD.1	Quantity of Comments	.2
SD.2	Effectiveness of Comments	.40
SD.3	Descriptiveness of Implementation Language	.50
MI.1	Machine Independence Measure	.2
CS.1	Procedure Consistency Checklist	.6
SI.1	Design Structure Measure	.75
SI.3	Complexity Measure	.23
SI.4	Code Simplicity Measure	.50
CO.1	Conciseness Measure	.90

NM - Not Measured

The values selected for Flexibility (.7) and Portability (.75) are above industry average since they were identified as very important for the AMT. The value for Maintainability (.7) was selected at what is considered the industry average because it was identified as important. The value for Reliability (.9) was identified as below the industry average since it was identified as only somewhat important to the AMT. The values we experienced on previous studies are also indicated in Table 5.2.2-3. These values were measured in a Command and Control (C^2) environment and a Management Information System (MIS) environment. Because of the nature of the C^2 application and the fact that the MIS system was a production system, the values for Reliability were higher for those systems than for the AMT.

The Metric values identified likewise were drawn from previous experience and depending on whether the quality factor they related to was considered important or not to the AMT, their values were set. The AMT scores as well as a discussion of the threshold values set and how they compare to previous experience is in paragraph 5.5.

5.2.3 APPLICATION METHODOLOGY

The procedures we used to apply the measurements to the AMT development are essentially those described in the Software Quality Measurement Manual [MCCJ78]. Those applied are briefly highlighted here:

(1) Established Quality Goals (see paragraph 5.2.2)

(2) Applied Worksheet 1 to the informal Requirements Specifications that were generated at the outset of the project.

(3) Applied Worksheet 2a to the Design Document at the system level.

(4) Applied Worksheet 2b to the Design Document at the module level. This application was made at the beginning of the implementation of each increment because it was at that point that the detailed designs of the modules in that increment were set.

(5) Applied Worksheet 3 to the code.

(6) Metric scores were calculated from the worksheets.

(7) Observations or analyses based on the worksheet data and the metric scores were documented (see paragraph 5.4).

(8) Where normalization functions existed, they were calculated.

(9) The worksheet data, metric scores, documented observations and quality ratings (calculated normalization function) are presented in this report.

(10) Where possible, automated tools were utilized to apply the metrics. Tools considered for use are identified in Appendix D.

5.2.4 DESIGN AND IMPLEMENTATION GUIDELINES

Based on the identification of certain quality factors as goals for the development, some specific practices were identified for the development team to follow. These guidelines are identified in Appendix B. These guidelines were derived from experiences in transferring PDP FORTRAN code to H6000 FORTRAN code, and from transferring code from the H6000 to an IBM 370.

5.3 APPLICATION OF WORKSHEETS

The worksheets that are part of the Software Quality Measurement Manual [MCCJ79] were the major vehicle for applying the measurements during the AMT development. Figure 3.3-1 illustrates the timing of their application. Note that worksheet 1 was applied to the draft Requirements Specification that was written in December 1979. That specification was not a formal deliverable of the contract. Worksheet 2a was applied to the Design Plan (CDRL A001).

Worksheet 2b was applied to the HIPO diagrams and Program Design Language description of each module in the Design Plan. The worksheets were applied at the initial phases of each subsystem as it was being developed. This timing was chosen because at that time the PDL's and data definitions were refined and were the driving documents of the implementation. It was at that time that identifying quality problems had the most positive impact.

Worksheet 3 was applied at the latter stages of each incremental development phase, when the source code was complete. Had an automated tool been available the measurements would have been taken several times during the implementation.

Worksheet 1 and 2a, as completed, are in Figures A-1 and A-2 and an example worksheet 2b and 3 are in Figure A-3 and A-4 in Appendix A. These latter two worksheets were completed at a module-level.

5.4 RESULTS

The results of the application of the worksheets were reported to RADC at two times during the project. The first time was during a review of the Design Plan at RADC. The second time was at the completion of the delivery of the AMT. The results were used by the development team thrughout the development effort to assess the quality of their design and implementation.

The results of the application of the worksheets were analyzed at three levels. First, some general observations were made based on the application of the worksheets. Second, the metric scores were calculated and reported. Third, the metric scores were compared with the quality goals for the project.

5.4.1 REQUIREMENTS AND DESIGN

5.4.1.1 General Observations At Requirements And Preliminary Design

Table 5.4.1.1-1 contains the general observations made based on the application of Worksheets 1 and 2a. These were applied to the draft requirements document prepared in the first month of the the contract and the design document, which was the product of the first phase of the contract and represents a system-level view. These worksheets and observations were made during the design phase of the contract and reported to the RADC and USACSC/AIRMICS project engineers at the design review.

The worksheet applications revealed that requirements dealing with such attributes as security, error tolerance, performance, and interfacing with other systems, had not been specified. Since security and performance were

Table 5.4.1.1-1 Observations Based on Worksheet Inspection of Requirements Specification and Preliminary Design Specification during Design Phase of the Project

<u>REQUIREMENTS</u> (Worksheet 1)

- No flow of processing and decisions during that flow described Operations concept did not really describe scenario of use
- No reliability requirements specified; error tolerance, error recovery
- No access controls required
- No discussion of user interface except for command language
- No performance requirements stated
- Provisions for interfacing with other systems lacking

<u>PRELIMINARY DESIGN SPEC</u> (Worksheet 2a)

- No error reporting/control system in effect
- Error conditions not identified yet
- No called/call matrix for modules yet
- No estimates on run times or storage requirements yet
- No access controls provided
- Other tools to interface with have not been identified
- User Manual not written yet (outline has been)
- No Test Plan yet

not major quality goals these were not considered for correction. Since Interoperability was important to the AMT, corrective action in the form of an analysis of how the AMT would interface with other tools was conducted. The results of the analysis were built into the design of the Data Management Subsystem.

The worksheet application to the preliminary design material revealed that a call/called matrix had not been generated at the subsystem level and that error handling in the system had not been described. Both of these deficiencies were corrected by the time the Design Plan (CDRL A001) was delivered. An update to worksheet 2a at detailed design was not done but one was done at the end of the contract. This update included the metrics related to the Test Plan and User's Manual. All worksheets have been delivered to RADC as part of the AMT data base and as a separate document.

5.4.1.2 Metric Scores

Table 5.4.1.2-1 contains the system metric scores calculated from the application of Worksheet 1 and 2a. Paragraph 5.4.1.3 contains an analysis of these scores. Only those metrics identified in paragraph 5.2 were measured.

5.4.1.3 Comparison with Quality Goals

The factors identified as very important and important were:

PORTABILITY:

The only indicator of portability at preliminary design time is the Modular Implementation measure (score of .57) which is average based on past experience. Measures of the machine independence and system independence are not made until detailed design. At the end of the preliminary design phase of the project the design was still machine and operating system independent.

FLEXIBILITY:

The Modular Implementation measure (.57) and the Generality of the design approach (.43) effect the flexibility of the code. These scores represent a slightly better than average score for flexibility according to past systems we had measured. While these are system level metrics

Table 5.4.1.2-1 Metric Scores from Initial Application of Worksheet 1 and 2a

REQUIRMENTS PHASE

Completeness (CP.1)	.8
Accuracy (AY.1)	0
Error Tolerance-Input Data (ET.2)	0
Error Tolerance-Computational Failures (ET.3)	0
Error Tolerance-Hardware Faults (ET.4)	0
Error Tolerance-Device Errors (ET.5	0
Access Control (AC.1)	.67
Access Audit (AA.1)	0
Operability (OP.1)	0
User Input Interface (CM.1)	1
User Output Interface (CM.2)	1
Communications Commonality (CC.1)	1
Data Commonality (DC.1)	1

PRELIMINARY DESIGN

Traceability (TR.1)	1
Completeness (CP.1)	.8
Accuracy (AY.1)	0
Error Tolerance-Control (ET,1)	0
Error Tolerance-Hardware Faults (ET.4)	0
Error Tolerance-Device Errors (ET.5	0
Design Structure (SI.1)	.33
Modular Implementation (MO.2)	.57
Generality (GE.1)	.43
Module Testing (IN.1)	Test Plan not completed yet
Integration Testing (IN.2)	
System Testing (IN.3)	
Iterative Processing Efficiency (EE.2)	0
Data Usage Efficiency (EE.3)	1
Storage Efficiency (SE.1)	0
Access Control (AC.1)	1
Access Audit (AA.1)	0
Operability (OP.1)	User manual not written yet
Training (TN.1)	
User Input Interface (CM.1)	
User Output Interface (CM.2)	
Communcation Commonality (CC.1)	.5
Data Commonality (DC.1)	1

if we substitute them into normalization equations we get a rating of approximately .25 or 4 man-days to make a modification to the system. This is better than the specified goal of 6 man-days to make a modification (rating = .7).

REUSABILITY:

At the preliminary design phase of the development the indicators available for reusability were the same as for flexibility.

INTEROPERABILITY:

The measures related to Interoperability are the Communication Commonality measure (score of 1) and Data Commonality measure (score of 1) during requirements and the same two (score of .5 and 1 respectively) plus Modular Implementation (.57) during preliminary design. The scores are high in this case because we had recognized the requirements to build a system with which it will be easy to interface. The primary interface will be with the data base. To interface a tool with AMT, one must write a translation or interface routine which takes the output of the tool and transforms it into the format of the AMT data base. The AMT data management routines would be available to facilitate that process.

CORRECTNESS:

The two measures which relate to completeness at requirements and preliminary design are the Completeness measure (.8) and the Traceability measure (1). The Consistency measures (1) were high because the design team agreed to a standard design notation.

MAINTENANCE:

The Design Structure measure (score of .33), the Modular Implementation measure (score of .57), and the consistency measures (1), relate to maintainability. The Design Structure measure was slightly lower than past experience has indicated it should be so we looked at it in some detail. Several modules were being called by many other modules at different levels of the system hierarchy. This lowered the design structure metric score. By identifying these modules as utilities the complexity of the design was decreased.

5.4.2 DETAILED DESIGN AND IMPLEMENTATION

5.4.2.1 General Observations At Detailed Design and Implementation

The difficulty in developing software remotely in terms of turnaround and obtaining complete output, and the fact that we had to apply the metrics manually to the AMT development, prevented as effective use of the metrics as would have been desired. Timeliness is especially critical during detailed design and implementation because in order to affect the design and implementation strategies the measurements have to be available on practically a daily basis. This was not possible. The metrics were applied once during the detailed design (worksheet 2b) and once when the code was complete (worksheet 3) and are reported here to provide assurance that a high quality product was provided. The information is also valuable for future extensions, modifications, and transporting of the AMT.

5.4.2.2 Metrics Scores

Table 5.4.2.2-1 provides a summary of the metric scores calculated from worksheets 2b and 3 applied to each module in the system. The scores shown are averaged over each subsystem and over the entire system. The system average score is calculated by taking the sum of each subsystem average score multiplied by the number of routines measured in that subsystem and dividing this sum by the total number of routines in the system. The measurements are taken from 58 modules representing over 12,000 lines of code. The breakdown of modules by subsystem is:

Executive Services Subsystems (EXS)	11
Automated Measurement Subsystem (AMS)	4
Data Management Subsystem (DMS)	12
Utilities Subsystem (UTL)	7
Report Generation Subsystem (RGS)	24

Not included in the measurements is the Preprocessing Subsystem (PPS) which includes the parser. This was existing code and the metrics were not applied to it during the AMT development.

Individual module scores were available through the Metrics Report and also through the Matrix Report. Analysis of these scores are in the following paragraph.

Table 5.4.2.2-1

Implementation Metric Scores

METRIC		SUBSYSTEM AVERAGE SCORES					SYSTEM
		AMS	EXS	UTL	DMS	RGS	AVERAGE
ACCURACY	AY.1	0	.27	.14	.17	.46	.29
CONCISENESS	CO.1	1.	1.	1.	1.	.99	.996
COMPLETENESS	CP.1	.25	.56	.38	.38	.75	.25
PROCEDURE CONSISTENCY	CS.1	.5	.73	.57	.58	.96	.76
DATA CONSISTENCY	CS.2	.25	.7	.29	.29	.48	.39
ITERATIVE PROCESSING EFFICIENCY	EE.2	.5	.55	.14	.92	.42	.51
DATA USAGE EFFICIENCY	EE.3	.46	.65	.45	.55	.67	.59
ERROR TOLERANCE CONTROL	ET.1	.75	.54	.43	.83	.96	.77
ERROR TOLERANCE INPUT DATA	ET.2	0	.38	.19	.17	.29	.25
ERROR TOLERANCE COMPUTATION	ET.3	.08	.15	.07	.12	.01	.07
DATA STORAGE EXPANDABILITY	EX.1	0	0	.14	.08	.21	.12
COMPUTATIONAL EXTENSIBILITY	EX.2	0	.05	.07	.08	.04	.05
IMPLEMENTATION GENERALITY	GE.2	.63	.73	.54	.83	.77	.74
MACHINE INDEPENDENCE	MI.1	.74	.88	.63	.84	.89	.84
MODULAR IMPLEMENTATION	MO.2	.31	.46	.38	.34	.36	.37
QUALITY OF COMMENTS	SD.1	.43	.37	.71	.59	.36	.46
EFFECTIVENESS OF COMMENTS	SD.2	.45	.60	.44	.58	.63	.58
DESCRIPTIVENESS OF LANGUAGE	SD.3	.75	.91	.71	.92	.96	.9
DESIGN STRUCTURE	SI.1	.45	.63	.45	.63	.63	.59
COMPLEXITY	SI.3	.09	.12	.02	.09	.17	.12
CODE SIMPLICITY	SI.4	.49	.62	.43	.69	.61	.6
SYSTEM SOFTWARE INDEPENDENCE	SS.1	.25	.36	.33	.29	.48	.38
TRACEABILITY	TR.1	0	.18	.14	0	.04	.07

5.4.2.3 Comparison With Quality Goals

Table 5.4.2.3-1 compares the subsystem and system average metric scores with the metric scores identified in Table 5.2.2-3. These metric scores were identified at the beginning of the project as goals the development team would attempt to meet. The values chosen, threshold values, were chosen because they represent either average or above average scores for those metrics based on past experience. They were not contractual requirements but were set as quality goals against which to assess the software development.

In most cases, the threshold values were met or exceeded providing some confidence in the quality of the software product. There were a few exceptions. The modular implementation metric (MO.2) is currently measured differently than previously specified. The modular implementation measure (MO.2) during past studies included the following measurements:

- Module size in lines of source code (1 if less than 100, 0 if greater than 100)
- Number of parameters which are control variables divided by number of total calling parameters.
- Input data controlled by calling module (1 if yes, 0 if no)
- Output data controlled by calling module (1 if yes, 0 if no)
- Control returned to calling modulue (1 if yes, 0 if no)
- Is temporary storage shared by call/called modules (1 if no, 0 if yes)

These measurements were added together and divided by six to get the metric value. Two additional measurements were added to the MO.2 metric in the AMT implementation. These were:

- 1 divided by number of elements passed as parameters that were not variables
- 1 divided by number of parameters not defined

The new metric value is the sum of the above six elements plus the new two measurements divided by eight. However, in taking these latter two measurements, the code inspectors interpreted both as zero when all parameters passed in a call statement were variables and all parameters were defined.

Table 5.4.2.3-1 Comparison of Metric Scores with Specified Thresholds

METRICS		SUBSYSTEM AVERAGE SCORES					SYS	SPECIFIED
		AMS	EXS	UTL	DMS	RGS	AVG	THRESHOLD
MODULAR IMPLEMENTATION	MO.2	.31	.46	.38	.34	.36	.37	.70
GENERAL CHECK LIST	GE.2	.63	.73	.54	.83	.77	.74	.35
QUANTITY OF COMMENTS	SD.1	.43	.37	.71	.59	.36	.46	.20
EFFECTIVENESS OF COMMENTS	SD.2	.45	.60	.44	.58	.63	.58	.40
DESCRIPTIVENESS OF LANGUAGE	SD.3	.75	.91	.71	.92	.96	.90	.50
MACHINE INDEPENDENCE	MI.1	.74	.88	.63	.84	.89	.84	.20
PROCEDURE CONSISTENCY	CS.1	.5	.73	.57	.58	.96	.76	.60
DESIGN STRUCTURE	SI.1	.45	.63	.45	.63	.63	.59	.75
COMPLEXITY	SI.3	.09	.12	.02	.09	.17	.12	.23
CODE SIMPLICITY	SI.4	.49	.62	.43	.69	.61	.60	.50
CONCISENESS	CO.1	.92	.94	.98	.91	.99	.94	.90

*These values were computed manually

This misinterpretation incorrectly lowered the MO.2 metric value by two eighths (2/8) or .25. Thus the MO.2 scores should have been .56, .71, .63, .59, .61 for the subsystems in Table 5.4.2.3-1 respectively and .62 for the system average. The scores still do not meet the threshold but average 89% of the threshold score.

The complexity measure did not meet the goal of .23 set. The logic of some of the routines to calculate the metric scores and to parse and identify the various constructs of COBOL is quite complicated. The average achieved, .12 is better that that recommended by McCabe [McCT] which equates to .1 in our metric. The design structure metric (SI.1) was slightly lower, .59 compared to the goal of .75, than the specified level.

To compare strictly using the system average is potentially misleading. The variation between subsystems is important to look at. A subsystem average may be quite low and in fact be a weak link, in terms of quality, within the system. The same analogy applies at a module level. The Exception Report provides the capability within AMT to identify those modules which are potential problem modules. The metric which varied greatest within the system was the complexity measure. This metric was used to monitor the development team and help during design and code walkthroughs to control the complexity of the design. Because the metric values were not contractual requirements, redesign and reimplementation was not done strictly to improve the metric value but only done when the complexity was obviously too high and would have a major impact on the quality of the software.

At the metric level, the AMT development team met 8 of the 11 (or 73%) of the specified goals.

Another view is illustrated in Table 5.4.2.3-2, where those metrics related to each quality factor and how well the software scored in terms of either the thresholds established or past experience is shown. In the case of metrics for which a threshold value was not established, the metric score was compared with past experience. The table identifies if the values achieved for the AMT were low (L), slightly higher (M), or much higher (H) than past experience.

Table 5.4.2.3-2 Metric Scores Related to Quality Goals

RELATED METRICS FACTORS	Modular Implementation MO.2	Quantity of Comments SD.1	Effectiveness of Comments SD.2	Descriptiveness of Language SD.3	System Software Independence SS.1	Machine Independence MI.1	Interface Measure GE.1	Implementation Generality GE.2	Data Storage Expandability EX.1	Computational Extensibility EX.2	Communication Commonality CC.1	Data Commonality DE.1	Traceability TR.1	Completeness CP.1	Procedure Consistency CS.2	Data Consistency CS.2	Design Structure SI.1	Complexity SI.3	Code Simplicity SI.4	Conciseness CO.1
PORTABILITY	N	Y	Y	Y	M	Y														
FLEXIBILITY	N	Y	Y	Y			NM	Y	M	M										
REUSEABILITY	N	Y	Y	Y	M	Y	NM	Y												
INTEROPERABILITY	N										NM	NM								
CORRECTNESS													L	L	Y	L				
MAINTAINABILITY	N	Y	Y	Y											Y	L	N	N	Y	Y

LEGEND

N	Threshold Value Not Achieved
Y	Threshold Value Achieved
L	Score Lower than Experience Base
M	Score Higher than Experience Base
H	Score Much Higher than Experience
NM	Not Measured

Using this view, the performance of the development team, in terms of the metrics relative to the six quality factors identified as important, can be assessed. For example, for the quality factor, portability, four metrics exceeded the specified threshold values, one metric for which a threshold was not specified, scored slightly better than past experience, and only one metric did not achieve the specified threshold value. Thus from a portability viewpoint, five out of six (83%) of the metrics related to portability exceeded expectations.

Using the metrics in this way, the development team could be assessed as having met 83% (5 of 6) of their goals related to portability, 86% (6 of 7) related to flexibility, 86% (6 of 7) related to reusability, 0% (0 of 1) for interoperability, 25% (1 of 4) for correctness, and 60% (6 of 10) for maintainability.

The NM indicator in the table identifies those metrics not measured. In most cases, these metrics were not measured because without automated support, it was not possible to measure them against any of the AMT source code.

In some cases, data or material was not available for measuring that particular metric. An example of this situation is the Data Communication (DC.1) metric. No other tool was identified to be interfaced with the AMT so no consideration was given to how compatible the AMT data structure was with any other tool.

Table 5.4.2.3-3 provides the results of substituting the average metric scores for the system into the normalization functions. These normalization functions, including the individual metric functions as well as the multivariate functions, are defined in the Software Quality Measurement Manual [MCCJ 79].

Because the Modular Implementation Measure (MO.2) was measured differently than previous studies, a constant of .25 was added to the MO.2 metric value to arrive at a corrected normalization function rating. This correction constant was only used for the normalization functions that contained the MO.2 metric.

Table 5.4.2.3-3 Normalization Function Performance For Implementation

		SCORE ACHIEVED	RATING	GOAL SET
PORTABILITY				
MULTIVARIATE FUNCTION	-1.7 + .19M (SD.1) +.76M (SD.2) + 2.5M (SD.3) + .64M (MI.1)	1.6*	1	.75
INDIVIDUAL FUNCTIONS	1.07M (SI.1)	.63		
	1.1M (MI.1)	.92		
	1.5M (SD.2)	.87		
FLEXABILITY				
MULTIVARIATE FUNCTION	.22M (MO.2) + .44M (GE.2) + .09M (SD.3)	.56	.91	.70
INDIVIDUAL FUNCTIONS	.6M (MO.2)	.37		
	.72M (GE.E)	.53		
	.59M (SD.2)	.34		
	56.M (SD.3)	.50		
MAINTAINABILITY				
MULTIVARIATE FUNCTION	-2 + 61M (SI.3) + 14M (MO.2) + .33M (SD.2)	.26	.62	.70
INDIVIDUAL FUNCTION	2.1M (SI.3)	.62		
	.71M (SD.2)	.41		
	.6M (SD.3)	.54		
	.5M (SI.1)	.30		
	.4M (SI.4)	.24		
RELIABILITY				
MULTIVARIATE FUNCTION	.48 M (ET.1) + .14 M (SI. 1)	.45	.9	.9
INDIVIDUAL FUNCTION	.57 M (ET.1)	.44		
	.58 M (SI.1)	.34		
	.53 M (SI.3)	.16		
	.53 M (SI.4)	.32		

*(See page 5-22)

In addition a correction to the normalization function for maintainability was made. This correction is to account for using a different complexity measure, SI.3. In the Factors in Software Quality study [MCCJ77a], Halstead's E measure [HALM77] was used. In the Metrics Enhancement [MCCJ79] and the AMT development, McCabe's [MCCT76] metric was used. This change was not taken into account in the normalization function documented in the Software Quality Measurement Manual. To account for the different measure, a factor of 2.46 multiplied by the complexity measure should be substituted for SI.3 in the normalization function shown in Table 5.4.2.3-3 to arrive at the calculated score. In the future, the normalization function recommended is:

$$- .2 + 1.5M(SI.3) + .14M(MO.2) + .33M(SD.2) = r_m$$

when using McCabe's metric as the complexity metric (SI.3).

The resultant ratings are compared with the established goals in Table 5.4.2.3-3. The individual factors are discussed below.

PORTABILITY

The AMT software is considered highly portable. The system software dependent and machine dependent software have been minimized and localized. The metric scores for these measures and others related to Portability are all relatively high except for the Modular Implementation Measure, MO.2.

The goal identified for portability was .75. The Software Quality Measurement Manual states that this rating is equivalent to 1 - (effort to transfer)/(effort to implement). Thus a goal of .75 is the same as saying the effort to transport a module of the AMT to another system should take 25% or less of the effort to implement that module. The score achieved by calculating the normalization function is equivalent to the rating. Thus a normalization function value of .9 equates to a rating of .9 and means the software can be transported to another system in 10% or less of the effort required to implement the software.

The multivariate function for portability and the individual normalization functions for all but the SI.1 metric exceeded the specified goal of .75. The portability of the AMT was demonstrated by the relative ease with which the initial prototype version of the AMT, developed on the VAX 11/780, was transported to the RADC H6180. The value of 1.6 should be interpreted as 1. Because the previous effort [McCJ79] to establish beta coefficients was based upon a limited sample of projects and the dependent variable of portability is somewhat illusive, the regression equations need to be adjusted to reflect a móre accurate representation. This will require additional portability data from a wider range of projects to be analyzed.

FLEXIBILITY

The AMT software exhibited characteristics that indicate it will be highly flexible. The metrics related to flexibility were all relatively high as shown in Table 5.4.2.3-1. A generalized parser was utilized in the Automated Measurement Services Subsystems to facilitate modifying the AMT to process other programming languages besides COBOL. The grammar description of COBOL developed was at a high enough level of abstraction to handle a wide number of COBOL grammars while still measuring the needed characteristics to calculate the metrics. The normalization function calculation resulted in a value of .56. This value relates to the average amount of effort it takes to make a modification to the software based on a change in requirements. The relationship is 1/.56 = the average person days to make a modification, or 1.78 person days. The rating for flexibility equals 1 - .05x(average person days to modify). The rating therefore is .91.

REUSEABILITY

The software exhibited high scores for those metrics related to reuseability. The interfaces and functional decomposition of the system are well defined to facilitate reuse. As shown in Table 5.4.2.3-1 and Table 5.4.2.3-2, six of the seven metrics related to reuseability exceeded expectations. These expectations were based on threshold values or past experience. In particular, the three metrics related to comments (DS.1, SD.2, SD.3), the machine independence metric (MI.1), and the implementation generality metric (GE.2), all exceeded the threshold values specified. The system software independence metric (SS.1) was higher than the values experienced during the Metric Enhancement study. The Modular Implementation metric (MO.2) was the only reuseability-related metric that did not meet or exceed the threshold value specified.

INTEROPERABILITY

While the metrics needed to assess this quality factor were not measured, facilites to allow interfacing with the AMT were built into the system. The PUT and GET commands can be utilized to interact with the AMT data base. They could be used to extract pertinent data from output of an existing software tools and placed into the AMT data base. The only metric measured that was related to interoperability was MO.2 which has been discussed already.

CORRECTNESS

The metrics related to this quality factor were mixed in their performance at best. Only one metric, CS.2, had a specified threshold value for the AMT development. That metric exceeded the threshold. Three other metrics related to correctness did not achieve values as high as those of past studies.

The interpretation that can be made is that the previous studies involved taking the measurements from existing operational systems which you would expect to be more mature, have more complete documentation, and therefore achieve higher metric scores than the AMT.

The testing process used five COBOL programs provided from a production system at the USACSC. The Test Plan (CDRL A0015) and the Test Analysis Report (CDRL A014) describe the test process. All planned tests except one were successfully accomplished. One functional capability not provided was the alternate print capability.

MAINTAINABILITY

The comments, structure, implementation techniques, and control flow complexity were controlled during the development, and these practices were reflected in the metric scores. The normalization function using the multiplication factor discussed previously for the complexity metric (SI.3) and the addition factor for the MO.2 metric resulted in a value of .26. This equates (1/.26) to an average of 3.8 person days to fix an error in the software. The rating then is 1 - .1x (average effort to fix an error) or .62. This is slightly lower than the .7 rating or goal specified. The complexity of the system was slightly higher than desired and resulted in the slightly lower rating.

RELIABILITY

Reliability was not a quality factor specified as critically important to the AMT because it is basically a prototype system. However, for evaluation purposes, we monitored the performance of the metric scores related to the reliability normalization function. The calculated normalization function value was .45. The rating is calculated by doubling this value to .9 and is equated to 1 - (number of errors)/(100 lines of code). The industry average is approximately 2 errors per 100 lines of code or .98. The .9 achieved by the AMT development met the goal specified.

5.5 COMPARISON OF AMT METRIC SCORES WITH PAST EXPERIENCES

Table 5.5-1 provides a comparison of the average metric scores for the AMT with past experiences. These past experiences include the JOVIAL Command and Control System used during the Factors in Software Quality Contract (MCCJ77), the Management Information System (MARDIS) written in COBOL and The Software Support System written in FORTRAN that were used in the Metrics Enhancement Contract (MCCJ79), a Data Base Management System written in JOVIAL, and a Telemetry Prediction Simulation System written in JOVIAL. The AMT was written in a structured FORTRAN. The annotation "NM" in the table indicates a metric that was not measured.

The following metrics had scores higher for the AMT than past experiences:

CO.1	Conciseness
ET.1	Error Tolerance
GE.2	Generality
MI.1	Machine Independence
SD.3	Descriptiveness of Language
SS.1	System Software Independence

These metrics indicate the concern primarily for portability and flexibility during the AMT development.

Table 5.5-1

Implementation Metric Score Comparisons

METRIC		AMT		COBOL		
		AVERAGE	JOVIAL	MIS	JOVIAL	JOVIAL
		SCORE	C2	FORTRAN	DBMS	EXS
ACCURACY	AY.1	.29	NM	NM	NM	NM
CONCISENESS	CO.1	.996	.78	.12	.75	.60
COMPLETENESS	CP.1	.25	.92	NM	NM	NM
PROCEDURE CONSISTENCY	CS.1	.76	.99	NM	NM	NM
DATA CONSISTENCY	CS.2	.39	.8	.68	NM	NM
ITERATIVE PROCESSING EFFICIENCY	EE.2	.51	.67	.50	NM	NM
DATA USAGE EFFICIENCY	EE.3	.59	.96	.85	NM	NM
ERROR TOLERANCE CONTROL	ET.1	.77	.77	.NM	NM	NM
ERROR TOLERANCE INPUT DATA	ET.2	.25	.84	.02	NM	NM
ERROR TOLERANCE COMPUTATION	ET.3	.07	.51	.07	NM	NM
DATA STORAGE EXPANDABILITY	EX.1	.12	NM	NM	NM	NM
COMPUTATIONAL EXTENSIBILITY	EX.2	.05	NM	.07	NM	NM
IMPLEMENATION GENERALITY	GE.2	.74	.48	.35	.12	.12
MACHINE INDEPENDENCE	MI.1	.84	.13	.21	NM	NM
MODULAR IMPLEMENTATION	MO.2	.37	.68	.71	NM	NM
QUANTITY OF COMMENTS	SD.1	.46	.69	.35	.38	.35
EFFECTIVE OF COMMENTS	SD.2	.58	.74	.40	NM	NM
DESCRIPTIVENESS OF LANGUAGE	SD.3	.90	.82	.57	NM	NM
DESIGN STRUCTURE	SI.1	.59	.64	.87	NM	NM
COMPLEXITY	SI.3	.12	.66	.23	.10	.08
CODE SIMPLICITY	SI.4	.60	.76	.57	.66	.73
SYSTEM SOFTWARE INDEPENDENCE	SS.1	.38	.18	.01	.03	.12
TRACEABILITY	TR.1	.07	1	NM	NM	NM
NORMALIZATION FUNCTION (ratings)						
PORTABILITY		1	NM	.23	NM	NM
FLEXIBILITY		.91	.88	.86	NM	NM
MAINTAINABILITY		.62	.68	.9	NM	NM
RELIABILITY		.9	.98	.96	NM	NM

NM = NOT MEASURED

The following metrics had scores lower for the AMT than past experiences:

CP.1	Completeness
CS.1	Procedure Consistency
CS.2	Data Consistency
EE.3	Data Usage Efficiency
MO.2	Modular Implementation
SI.1	Design Structure
TR.1	Traceability

These metrics indicate a lesser attention provided to characteristics related to correctness and reliability. The scores of the AMT metrics were not low in the absolute sense but were lower than those achieved in the command and control software and other contract deliverable software. This is understandable considering the AMT is a prototype research tool. Also shown in the table are the ratings achieved for the four factors that have established normalization functions. The relative ratings for the Factors in Software Quality (FSQ) study, the Metrics Enhancement (ME) study, and the Automated Measurement Tool (AMT) development for these factors were (from high to low):

PORTABILITY	MAINTAINABILITY
AMT	ME
FSQ	FSQ
ME	AMT

FLEXIBILITY	RELIABILITY
AMT	FSQ
FSQ	ME
ME	AMT

5.6 EXPERIMENT CONCLUSIONS

As a result of applying the metrics during the development of the AMT several general observations can be made. First the use of the quality factors to identify what qualities were desired provided an excellent technique for focusing standards and conventions and the goals of the development team to meet the customers requirements. Second, the use of the metrics during the development as a development team tool as well as a mechanism for reviewing requirements with the customer proved effective. Third, based on the metrics, the AMT development was reasonably successful at achieving the quality goals set at the beginning of the project. At the metric level, 8 of 11 (83%) specified goals were met. Of the three metric thresholds not achieved, the scores realized were 89%, 79% and 52% of the values desired. At the normalization functon level, 3 of 4 specified goals were met. The one not met (maintainability) was 89% of the desired value.

There were some negative aspects identified. First, the setting of the specific quality goals was done with relatively little experience data. In some cases, such as the modular implementation (MO.2) where the metric algorithm changed and the goal had been set too high, the goals established were not reasonable. The setting of goals should be carefully considered and reviewed between the customer and development team. Second, continued validation of the normalization functions is required. A complete validation, i.e., statistical analysis, of the data should be performed on new sets of data to gain more confidence in the normalization functions accuracy. Third, considerable interaction between the customer and the development team is needed to ensure effective use of the quality feedback provided by the metrics. Tradeoff analyses are necessary to ensure wasted effort is not spent correcting deficiencies which are not important or measuring metrics which are not critical. Fourth, automated support was not available and hindered the effective daily use of the metrics by the development team. In general the following conclusions can be drawn:

(1) The metrics proved to be an effective tool for setting quality goals, identifying standards and conventions to guide the development, and monitoring the progress toward these goals in-line with the development.

(2) Automated tools are necessary to provided reliable, timely metric information.

(3) An interactive customer is necessary. More quantitative information about the software product is available and should be used.

(4) The normalization functions need continued validation before they can be generally used. They should be validated and tailored to specific applications and development environments.

(5) The metrics could be utilized as a contractual instrument. The recommendation is to use them for determining incentive or award fees. Their use as an absolute acceptance criteria is possible but the specific metrics and threshold values would have to be negotiated prior to contract start.

SECTION 6
FUTURE DEVELOPMENT

The AMT was developed to demonstrate the concept of automated collection and reporting of software metrics. A minimum set of metrics are automatically collected from COBOL source code. A fairly extensive set of reports are generated to fulfill the requirements of a number of personnel who might use the AMT.

Several areas of the AMT could be enhanced for use on an actual large scale software development. Under the category of enhancements, the following aspects of the AMT could be modified or added:

(1) Add a form entry system for easier manual input of worksheet data.

(2) Modify the Report Generation Services Subsystem to be more flexible in providing user defined reports.

(3) Provide an interface to a statistical package.

(4) Interface AMT with other tools, especially tools that would support automated measurement during requirements definition and design phases.

(5) Expand COBOL grammar description and Automated Measurement Services Subsystem to support additional metrics automation.

(6) Define another language grammar (eg. FORTRAN) to parser, develop scanner and incorporate processing capability for another language.

(7) Tie AMT into Configuration Control and Error Reporting Systems or Program Support Libraries.

(8) Transport AMT to other computing environments.

(9) Expand data base capabilities beyond 50 modules.

SECTION 7
REFERENCES

[ALJM79] Al-Jarrah, M., et al
"An Empirical Analysis of COBOL Programs"
Software - Practice and Experience, Vol. 9, Issue No.5, May 1979.

[BASV78] Basili, V., et al
"Investigating Software Development Approaches"
AFOSR TR-688, August 1978.

[BAUF73] Bauer, F. L. (Ed)
Advanced Course on Software Engineering
Springer - Verlag, Berline, 1973.

[BOEB73] Boehm, B.
"Software and Its Impact: A Quantitative Report"
Datamation, April 1973.

[CAVJ78] Cavano, J., McCall, J.
"A Framework for the Measurement of Software Quality",
Proceedings ACM Software Quality Assurance Workshop, November 1978.

[CHER] Chevance, R. J., et al
"Static Profile and Dynamic Behavior of COBOL Programs"
SIGPLAN, reference open.

[CONS75] Constantine, L. Yourdon, E.
Structured Design, Yourdon Press, N. Y., 1975.

[CULK79] Culik
"The Cyclomatic Number and the Normal Number of Programs"
ACM SIGPLAN Notices, Vol. 14, No. 4, April 1979.

[DeMR76] DeMille, R. A., et al
"Can Structured Programs be Efficient?", ACM SIGPLAN Notices,
October 1976.

[DEWR78] Dewar, R., Hage, J.
"Size, Technology, Complexity, and Structual Differentiation: Toward a Theoretical Synthesis", Adminstrative Science Quarterly, pp 111-136, March 1978.

[DIJE69] Dijkstra, E. W.
"NATO Science Committee Report", January 1969.

[DoDMAN] DoD Manual 4120.17-M
Automated Data Systems Documentation Standards

[DZIW78] Dzida, W., et al
"User-Perceived Quality of Interactive Systems", Proceedings of 3rd International Conference on Software Engineering, 1978

[FAGM76] Fagan, M. E.
"Design and Code Inspections and Process Control in the Development of Programs", IBM Technical Report TR 00.2763, Poughkeepsie, 1976.

[FITA78] Fitzsimmons, A, Love, T.
"A Review and Evaluation of Software Science",
ACM Commuting Surveys, Vol. 10, No. 1, March 1978.

[FLEJ72] Fleiss, J. E., et al
"Programming for Transferability"
NTIS Memorandum AD-750 897, 1972.

[FOSL76] Fosdick, L. D., Osterweil, L. J.
"Data Flow Analysis in Software Reliability", ACM Computing Surveys Special Issue: Reliable Software I, 1976.

[FRIR78] Fried, R.
"Monitoring Data Integrity"
Datamation, June 1978.

[GAIE78] Gainer, E., et al
"The Design of a Reliable Application System"
Proceedings of the 3rd International Conference on Software Engineering,
1978.

[GELD79] Gelperin, D.
"Testing Maintainability"
ACM Software Engineering Notes, Vol. 4, No. 2, April 1979.

[GOLJ73] Goldberg, J., ed.
Proceedings of the Symposium on the High Cost of Software,
Monterey, 1973

[GORG71] Gorry, G. A., Scott Morton, M.S.
"A Framework for Management Information Systems"
Sloan Management Review, Vol. 13, No. 1,
Fall 1971, MIT Cambridge, Mass.

[HALM77] Halstead, M.
Elements of Software Science Elseview Computer Science Library,
New York, 1977.

[HANS76] Hantler, S. L., King, J. C.
"An Introduction to Proving the Correctness of Programs"
ACM Computing Surveys Special Issue: Reliable Software I,
September 1976.

[HECS77] Hecht, M. S.
Flow Analysis of Computer Programs, Elsevier North-Holland,
New York, 1977.

[HETB78] Hetzel, B.
"A Perspective on Software Development"
Proceedings of the 3rd International Conference on Software Engineering,
1978.

[HOAC78] Hoare, C.A.R.
"Software Engineering: A Keynote Address", Proceedings of the 3rd International Conference on Software Engineering, 1978.

[HORJ73] Horning, J. J., Randell, B.
"Process Structuring"
ACM Computing Surveys, Vol. 5, No. 1, March 1973.

[IMP74] "Improved Programming Technologies - An Overview"
IBM TR-GC20-1850-0, 1974.

[JACM78] Jackson, M. A.
"Information Systems: Modeling, Sequences and Transformation"
Proceedings of the 3rd International Conference on Software Engineering, 1978.

[JOHJ75] Johnson, J. P.
"Software Reliability Measurement"
NTIS AD-A019-147, December 1975.

[KAUR75] Kauffman, R.
"COBOL/Structured Programming - Will the Marriage Survive"
Infosystems February 1975.

[KNUD73] Knuth, D. E.
"A Review of "Structured Programming",
STAN-CS-73-371 Computer Science Dept., Stanford University, 1973.

[KOSS74] Kosaraju, S. R., Ledgard, M. F.
Concepts in Quality Software Design
NBS Technical Note 942, Washington 1974.

[KURS75] Kurki-Suonio, R.
"Towards Better Structured Definitions of Programming Languages",
STAN-CS-75-500 Computer Science Dept., Stanford University, 1975.

[LIEB78] Lientz, B., et al
"Characteristics of Applications Software Maintenance"
Communications of the ACM, Vol. 21, No. 6, June 1978.

[MATM78] Matsumoto, M.
"Design and Quality in MIS Environments"
Software Metrics Enhancement Task Internal Memorandum No. 1,
August 1978.

[McCC78] McClure, C. L.
Reducing COBOL Complexity through Structured Programming
Van Nostrand Reinhold Co., 1978.

[McCJ77a] McCall, J., Richards, P., Walters, G.
"Factors in Software Quality", 3 Vols.
RADC TR 77-369, November 1977.

[McCJ77b] McCall, J., Richards, P., Walters, G.
"Metrics for Software Quality Evaluation and Prediction"
Proceedings of the NASA/Goddard Second Summer Engineering Workshop,
September 1977.

[McCJ78a] McCall, J.
"The Quality of Software Quality Metrics in Large-Scale Software Systems Development", Proceedings of the Second Software Life Cycle Management Workshop, August 1978.

[McCJ79] McCall, J., Matsumoto, M.
"Software Quality Metrics Enhancements"
RADC TR 80-109, April 1980.

[McCJ78b] McCall, J.
"Software Quality: The Illusive Measurement"
Software Quality Management Conference, September 1978.

[MCCT76] McCabe, T. J.
"A Complexity Measure", IEEE Transactions on Software Engineering,
December, 1976.

[MCKJ79] McKissick, J., et al
"The Software Development Notebook - A Proven Technique" Proceedings 1979 Annual Reliability and Maintainability Symposium, January 1979.

[MILE79] Miller, E.
"Some Statistics from the Software Test Factory"
ACM Software Engineering Notes, Vol. 4, No. 1, January 1979.

[LITB78] Littlewood, B.
"How to Measure Reliability, and How Not to..."
3rd Proceedings of the International Conference on Software Engineering, Atlanta, 1978.

[LOVL77] Love, L. T.
Relating Individual Difference in Computer Programming Performance to Human Information Processing Abilities, Ph.D Thesis University of Washington, 1977.

[LOVT77A] Love, T.
"An Experimental Investigation of the Effect of Program Structure on Program Understanding", G.E. Technical Information Series TIS77ISP006, 1977.

[LOVT776] Love, T.
"A Preliminary Experiment to Test Influence on Human Understanding of Software", G.E. Technical Information Series TIS77ISP007, 1977.

[LUCH741 Lucas, H. C.
Toward Creative Systems Design
Columbia University Press, New York, 1974

[LYOG78] Lyon, G.
"COBOL Instrumentation and Debugging: A Case Study" NBS Special Publication 500-26, U.S. Dept. of Commerce 1978.

[MILSTD] MIL-STD-490
Specification Practices

[MIYI78] Miyamoto, I.
"Towards an Effective Software Reliability Evaluation" Proceedings of the 3rd International Conference on Software Engineering, 1978.

[MYEG75] MYERS, G. S.
Reliable Software Through Composite Design
Petrocelli/Charter, 1975.

[PAND76] Panzl, D. J.
"Test Procedures: A New Approach to Software Verification" Proceedings of the Second International Conference on Software Engineering, San Francisco, 1976.

[PARD75] Parnas, D. L.
"The Influence of Software Structure on Reliability", Proceedings of the International Conference on Reliable Software, Los Angeles, 1975.

[PEDJ78] Pederson, J. T., Buckle, J. K.
"Kongsberg's Road to an Industrial Software Methodology", Proceedings of the 3rd Internation Conference on Software Engineering, 1978.

[PYSA78] Pyster, A., Dutra, A.
"Error-Checking Compilers and Portability"
Software Practice and Experience, Vol. 8, Issue 1,
January - February 1978.

[RICP76] Richards, P., Chang, P.
"Localization of Variables: A Measure of Complexity", GE TIS 76CIS07, December 1976.

[RIDW78] Riddle, W. E., et al
"Behavior Modelling During Software Design"
Proceedings of the 3rd International Conference on Software Engineering, 1978.

[ROBL75] Robinson, L., et al
"The Verification of COBOL Programs"
NTIS Memorandum, June 1975.

[SAMS76] "Contractor Software Quality Assurance Evaluation Guide"
SAMSO Pamphlet 74-2, Los Angeles, 1976.

[STR74] "Structured Programming Series"
RADC, 15 Vols., 1974-1975.

[TAGW77] Taggart, W. M. Jr, Tharp, M. O.
"A survey of Information Requirements Analysis Techniques" ACM Computing Surveys, Vol. 9, No. 4, 1977.

[USACSCM] USACSC Manual 18-1
Automatic Data Processing System Development, Maintenance and Documentation Standards and Procedures Manual.

[VINW77] Vinson, W. D., Heany, D. F.
"Is Quality Out of Control?"
Harvard Business Review, November-December 1977.

[WALG78a] Walter, G., McCall. J.
"The Development of Metrics for Software R&D"
1978 Proceedings, Annual Reliability and Maintainability Symposium, January 1978.

[WALG78b] Walters, G.
"Application of Metrics to Software Quality Management Programs", Software Quality Management Conference, September 1978.

[WEGP76] Wegner, P.
"Research Paradigms in Computer Science"
Proceedings of the 2nd International Conference on Software Engineering, San Francisco, 1976.

[WEGP78] Wegner, P.
"Research Directions in Software Technology"
Proceedings of the 3rd International Conference on Software Engineering,
1978.

[WIRN65] Wirth, N.
"On Certain Basic Concepts of Programming Languages"
Technical Report No, CS65, Computer Science Department, Stanford University, 1965.

[WONG78] Wong, G.
"Design Methodology for Computer System Modeling Tools"
Symposium on Modeling and Simulation Methodology,
August 1978, Rehorot, Isreal.

[YEHR76] Yeh, R. T., ed.
"Software Validation", ACM Computing Surveys, Special Issue; Reliable Software I, 1976.

RECOMMENDED REFERENCES

VAX/VMS Command Language User's Guide - Order No. AA-D023B-TE

VAX-11 FORTRAN IV-PLUS Language Reference Manual - Order No. AA-D034A-TE

VAX-11 FORTRAN IV-PLUS User's Guide - Order No. AA-D035A-TE

Honeywell TSS General Information Manual - Series 60 (Level 66)/6000
Order No. DD22

Honeywell FORTRAN Reference Manual - Series 60 (Level 66)/6000
Order No. DG75

General Electric NED Time-Share User's Guide NEDE-21328 Class II

APPENDIX A
SAMPLE REPORTS

Pg. 1

METRIC WORKSHEET 1 REQUIREMENTS ANALYSIS/SYSTEM LEVEL	SYSTEM NAME: AMT	DATE 12-77 INSPECTOR: MATSUMOTO

I. COMPLETENESS (CORRECTNESS, RELIABILITY)

1. Number of major functions identified (equivalent to CPCI). CP.1 — 5
2. Are requirements itemized so that the various functions to be performed, their inputs and outputs, are clearly delineated? CP.1(1) — [✓ Y] N
3. Number of major data references. CP.1(2) — 33
4. How many of these data references are not defined? CP.1(2) — 0
5. How many defined functions are not used? CP.1(3) — 0
6. How many referenced functions are not defined? CP.1(4) — 0
7. How many data references are not used? CP.1(2) — 0
8. How many referenced data references are not defined? CP.1(6) — 0
9. Is the flow of processing and all decision points in that flow described? CP.1(5) — Y N
10. How many problem reports related to the requirements have been recorded? CP.1(7) — 0
11. How many of those problem reports have been closed (resolved)? CP.1(7) — N/A

II. PRECISION (RELIABILITY)

1. Has an error analysis been performed and budgeted to functions? AY.1(1) — Y [✓ N]
2. Are there definitive statements of the accuracy requirements for inputs, outputs, processing, and constants? AY.1(2) — Y [✓ N]
3. Are there definitive statements of the error tolerance of input data? ET.2(1) — Y [✓ N]
4. Are there definitive statements of the requirements for recovery from computational failures? ET.3(1) — Y [✓ N]
5. Is there a definitive statement of the requirement for recovery from hardware faults? ET.4(1) — Y [✓ N]
6. Is there a definitive statement of the requirements for recovery from device errors? ET.5(1) — Y [✓ N]

III. SECURITY (INTEGRITY)

1. Is there a definitive statement of the requirements for user input/output access controls? AC.1(1) — [✓ Y] N
2. Is there a definitive statement of the requirements for data base access controls? AC.1(2) — [✓ Y] N
3. Is there a definitive statement of the requirements for memory protection across tasks? AC.1(3) — Y [✓ N]
4. Is there a definitive statement of the requirements for recording and reporting access to system? AA.1(1) — Y [✓ N]
5. Is there a definitive statement of the requirements for immediate indication of access violation? AA.1(2) — Y [✓ N]

METRIC WORKSHEET 1 REQUIREMENTS ANALYSIS/SYSTEM LEVEL	SYSTEM NAME: ________	DATE ________ INSPECTOR: ________

IV. HUMAN INTERFACE (USABILITY)

1. Are all steps in the operation described (operations concept)? OP.1(1) [✓ Y] N
2. Are all error conditions to be reported to operator/user identified and the responses described? OP.1(2) [✓ Y] N
3. Is there a statement of the requirement for the capability to interrupt operation, obtain status, modify, and continue processing? OP.1(3) [✓ Y] N
4. Is there a definitive statement of requirements for optional input media? CM.1(6) [✓ Y] N
5. Is there a definitive statement of requirements for optional output media? CM.2(7) [✓ Y] N
6. Is there a definitive statement of requirements for selective output control? CM.2(1) [✓ Y] N

V. PERFORMANCE (EFFICIENCY)

1. Have performance requirements (storage and run time) been identified for the functions to be performed? EE.1 Y [✓ N]

VI. SYSTEM INTERFACES (INTEROPERABILITY)

1. Is there a definitive statement of the requirements for communication with other systems? CC.1(1) Y [✓ N]
2. Is there a definitive statement of the requirements for standard data representations for communication with other systems? DC.1(1) Y [✓ N]

VII. INSPECTOR'S COMMENTS

Make any general or specific comments that relate to the quality observed while applying this checklist.

Pg. 1

METRIC WORKSHEET 2A DESIGN/SYSTEM LEVEL	SYSTEM NAME: AMT	DATE: INSPECTOR: Stone

I. COMPLETENESS (CORRECTNESS, RELIABILITY)

1. Is there a matrix relating itemized requirements to modules which implement those requirements? TR.1 — [✓ Y] N
2. How many major functions (CPCIS) are identified? CP.1 — NA
3. How many functions identified are not defined? CP.1(2) — 0
4. How many defined functions are not used? CP.1(3) — 0
5. How many interfaces between functions are not defined? CP.1(6) — 1
6. Number of total problem reports recorded? CP.1(7) — 0
7. Number of those reports that have not been closed (resolved?) CP.1(7) — 0

Profile of problem reports: (number of following types)

8. Computational
9 Logic
10. Input/output
11. Data handling
12. OS/System Support
13. Configuration
. . Routine/Routine interface
15. Routine/System Interface
16. Tape Processing
17. User interface
18. data base interface
19. user requested changes
20. Preset data
21. Global variable definition
22. Recurrent errors
23. Documentation
24. Requirement compliance
25. Operator
26. Questions
27. Hardware

II. PRECISION (RELIABILITY)

1. Have math library routines to be used been checked for sufficiency with regards to accuracy requirements? AY.1(3) — Y [✓ N]
2. Is concurrent processing centrally controlled? ET.1(1) — Y [✓ N]
3. How many error conditions are reported by the system? ET.1(2) — Y [✓ N]
4. How many of those errors are automatically fixed or bypassed and processing continues? ET.1(2) — 0
5. How many, require operator intervention? ET.1(2) — 0
6. Are provisions for recovery from hardware faults provided? ET.4(2) — Y [✓ N]
7. Are provisions for recovery from device errors provided? ET.5(2) — Y [✓ N]

III STRUCTURE (RELIABILITY, MAINTAINABILITY, TESABILITY, PORTABILITY, REUSABILITY, INTEROPERABILITY)

1. Is a hierarchy of system, identifying all modules in the system provided? SI.1(1) — [✓ Y] N
2. Number of Modules SI.1(2) MO.2(1) — 114
3. Are there any duplicate functions? SI.1(2) — Y [✓ N]
4. Based on hierarchy or a call/called matrix, how many modules are called by more than one other module? GE.1 MO.2(1) — 83
5. Are the constants used in the system defined once? GE.2(5) — [✓ Y] N

METRIC WORKSHEET 2A DESIGN/SYSTEM LEVEL	SYSTEM NAME ________	DATE: ________ INSPECTOR: ________

IV. OPTIMIZATION (EFFICIENCY)

1. Are storage requirements allocated to design? SE.1(1)	[✓ Y]	N
2. Are virtual storage facilities used? SE.1(2)	[✓ Y]	N
3. Is dynamic memory management used? SE.1(5)	[✓ Y]	N
4. Is a performance optimizing compiler used? EE.2(2)	Y	[✓ N]
5. Is global data defined once? CS.2(3)	[✓ Y]	N
6. Have Data Base or files been organized for efficient processing? EE.3(5)	[✓ Y]	N
7. Is data packing used? EE.2(5)	Y	[✓ N]
8. Number of overlays EE.2(4)	0	
9. Overlay efficiency - memory allocation EE.2(8)	NA	
10. max overlay size		
11. min overlay size		

V. SECURITY (INTEGRITY)

1. Are user Input/Output access controls provided? AC.1(1)	[✓ Y]	N
2. Are Data Base access controls provided? AC.1(2)	[✓ Y]	N
3. Is memory protection across tasks provided? AC.1(3)	[✓ Y]	N
4. Are there provisions for recording and reporting errors? AC.2(1,2)	[✓ Y]	N

VI. SYSTEM INTERFACES (INTEROPERABILITY)

1. How many other systems will this system interface with? CC.1(1)	1	
2. Have protoc 1 standards been established? CC.1(2)	[✓ Y]	N
3. Are they being complied with? CC.1(2)	[✓ Y]	N
4. Number of modules used for input and output to other systems? CC.1(3,4)		1
5. Has a standard data representation been established or translation standards between representations been established? DC.1(1)	[✓ Y]	N
6. Are they being complied with? DC.1(2)	[✓ Y]	N
7. Number of modules used to perform translations? DC.1(3)	1	

VII. HUMAN INTERFACE (USABILITY)

1. Are all steps in operation described including alternative flows? OP.1(1)	Y	N
2. Number of operator actions? OP.1(4) ?	NA	

METRIC WORKSHEET 2A DESIGN/SYSTEM LEVEL	SYSTEM NAME:____________	DATE:____________ INSPECTOR____________

VII. HUMAN INTERFACE (USABILITY) (Continued) ?

3.	Estimated or Actual time to perform? OP.1(4)		
4.	Budgeted time for complete job? OP.1(4)		
5.	Are job set up and tear down procedures described? OP.1(5)	Y	N
6.	Is a hard copy of operator interactions to be maintained? OP.1(6)	Y	N
7.	Number of operator messages and responses? OP.1(2)		
8.	Number of different formats? OP.1(2)		
9.	Are all error conditions and responses appropriately described? OP.1(2)	Y	N
10.	Does the capability exist for the operator to interrupt, obtain status, save, modify, and continue processing? OP.1(3)	Y	N
11.	Are lesson plans/training materials for operators, end users, and maintainers provided? TN.1(1)	Y	N
12.	Are realistic, simulated exercises provided? TN.1(2)	Y	N
13.	Are help and diagnostic information available? TN.1(3)	Y	N
14.	Number of input formats CM.1(2)		
15.	Number of input values CM.1(1)		
16.	Number of default values CM.1(1)		
17.	Number of self-identifying input values CM.1(3)		
18.	Can input be verified by user prior to execution? CM.1(4)	Y	N
19.	Is input terminated by explicitly defined by logical end of input? CM.1(5	Y	N
20.	Can input be specified from different media? CM.1(6)	Y	N
21.	Are there selective output controls? CM.2(1)	Y	N
22.	Do outputs have unique descriptive user oriented labels? CM.2(5)	Y	N
23.	Do outputs have user oriented units? CM.2(3)	Y	N
24.	Number of output formats? CM.2(4)		
25.	Are logical groups of output separated for user emamination? CM.2(5)	Y	N
26.	Are relationships between error messages and outputs unambiguous? CM.2(6)	Y	N
27.	Are there provisions for directing output to different media? CM.2(7)	Y	N

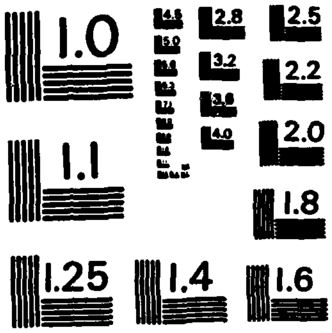
VIII. TESTING (TESTABILITY) APPLY TO TEST PLAN, PROCEDURES, RESULTS ?

1. Number of paths? IN.1(1) ______
2. Number of paths to be tested? IN.1(1) ______
3. Number of input parameters? IN.1(1) ______
4. Number of input parameters to be tested? IN.1(2) ______
5. Number of interfaces? IN.2(1) ______

AD-A121 360   AUTOMATION OF QUALITY MEASUREMENT(U) GENERAL ELECTRIC   2/2
CO UUNNYVALE CALIF J A MCCALL ET AL. SEP 82
RADC-TR-82-247 F30602-79-C-0267

UNCLASSIFIED   F/G 9/2   NL

END
FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

METRIC WORKSHEET 2A DESIGN/SYSTEM LEVEL	SYSTEM NAME; ____________	DATE: INSPECTOR;____________

VIII. TESTING (TESTABILITY) - APPLY TO TEST PLAN, PROCEDURES, RESULTS (CONTINUED) 7

6. Number of interfaces to be tested? IN.2(1)		9. Number of modules? IN.3(1)		
7. Number of itemized performance requirements? IN.2(2)		10. Number of modules to be exercised? IN.3(1)		
8. Number of performance requirements to be tested? IN.2(2)		11. Are test inputs and outputs provided in summary form? IN.3(2)	Y	N

IX DATA BASE

1. Number of unique data items in data base SI.1(6)	296
2. Number of preset data items SI.1(6)	12
3. Number of major segments (files) in data base SI.1(7)	3

X INSPECTOR'S COMMENTS

Make any general or specific comments about the quality observed while applying this checklist.

METRIC WORKSHEET 2B
DESIGN/MODULE LEVEL

SYSTEM NAME: AMT
MODULE NAME: AMSPUT

DATE: 1 July 81
INSPECTOR: MCGINDLEY

I. COMPLETENESS (CORRECTNESS, RELIABILITY)

1. Can you clearly distinguish inputs, outputs, and the function being performed? CP.1(1) [✓ Y]

2. How many data references are not defined, computed, or obtained from an external source? CP.1(2) Ø

3. Are all conditions and processing defined for each decision point? CP.1(5) [✓ Y]

4. How many problem reports have been recorded for this module? CP.1(7) Ø

Profile of Problem Reports: ⇒

4. Number of problem reports still outstanding CP.1(7) Ø

5. Computational	NA
6. Logic	↓
7. Input/Output	
8. System/OS Support	
9. Configuration	
10. Routine/Routine Interface	
11. Routine/System Interface	
12. Tape Processing	
13. User Interface	
14. Data Base Interface	
15. User Requested Changes	
16. Preset Data	
17. Global Variable Definition	
18. Recurrent Errors	
19. Documentation	
20. Requirement Compliance	
21. Operator	
22. Questions	
23. Hardware	

II. PRECISION (RELIABILITY)

1. When an error condition is detected, is it passed to calling module? ET.1(3) [✓ Y] N

2. Have numerical techniques being used in algorithm been analyzed with regards to accuracy requirements? AY.1(4) Y [✓ N]

3. Are values of inputs range tested? ET.2(2) Y [✓ N]

- Are conflicting requests and illegal combinations identified and checked? ET.2(3) Y [✓ N]

5. Is there a check to see if all necessary data is available before processing begins? ET.2(5) Y [✓ N]

6. Is all input checked, reporting all errors, before processing begins? ET.2(4) Y [✓ N]

7. Are loop and multiple transfer index parameters range tested before use? ET.3(2) Y [✓ N]

8. Are subscripts range tested before use? ET.3(3) Y [✓ N]

9. Are outputs checked for reasonableness before processing continues? ET.3(4) Y [✓ N]

III. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY)

1. How many Decision Points are there? SI.3 5

2. How many subdecision Points are there? SI.3 Ø

3. How many conditional branches are there? SI.3 5

4. How many unconditional branches are there? SI.3 Ø

METRIC WORKSHEET 2B DESIGN/MODULE LEVEL	SYSTEM NAME: AMT MODULE NAME: AMSPUT	Pg. 2

III. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY) (CONTINUED)

5. Is the module dependent on the source of the input or the destination of the output? SI.1(2)	Y [✓ N]
6. Is the module dependent on knowledge of prior processing? SI.1(3)	[✓ Y] N
7. Are any limitations of the processing performed by the module identified? EX.2(1)	Y
8. Number of entrances into modules SI.1(5)	1
9. Number of exits from module SI.1(5)	1

IV. REFERENCES (MAINTAINABILITY, FLEXIBILITY, TESTABILITY, PORTABILITY, REUSABILITY, INTEROPERABILITY)

1. Number of references to system library routines, utilities or other system provided facilities SS.1(1)	Ø
2. Number of input/output actions MI.1(2)	3
3. Number of calling sequence parameters MO.2(3)	1
4. How many calling sequence parameters are control variables MO.2(3)	Ø
5. Is input passed as calling sequence parameters MO.2(4)	Y [✓ N]
6. Is output passed back to calling module? MO.2(5)	[✓ Y] N
7. Is control returned to calling module MO.2(6)	[✓ Y] N
8. Is temporary storage shared with other modules? MO.2(7)	[✓ N]
9. Does the module mix input, output and processing functions in same module? GE.2(1)	Y
10. Number of machine dependent functions performed? GE.2(2)	Ø
11. Is processing data volume limited? GE.2(3)	[✓ Y]
12. Is processing data value limited? GE.2(4)	[✓ Y]
13. Is a common, standard subset of programming language to be used? SS.1(2)	[✓ Y]
14. Is the programming language available in other machines? MI.1(1)	[✓ Y]

V. EXPANDABILITY (FLEXIBILITY)

1. Is logical processing independent of storage specification? EX.1(1)	Y
2. Are accuracy, convergence, or timing attributes parametric? E	Y
3. Is module table driven? EX.2(2)	Y

VI. OPTIMIZATION (EFFICIENCY)

1. Are specific performance requirements (storage and runtime) allocated to this module? EE.1	Y

METRIC WORKSHEET 2B DESIGN/MODULE LEVEL	SYSTEM NAME: AMT MODULE NAME: AMSPJT	Pg. 3

OPTIMIZATION (EFFICIENCY) (CONTINUED)

2. Which category does processing fall in: EE.2

Circle one

1 Real-time

2 On-line

[✓ 3] Time-constrained

4 Non-time critical

3. Are non-loop dependent functions kept out of loops? EE.2(1)	Yes
4. Is bit/byte packing/unpacking performed in loops? EE.2(5)	No
5. Is data indexed or reference efficiently? EE.3(5)	Yes

VII. FUNCTIONAL CATEGORIZATION

Categorize function performed by this module according to following: Circle one below

1 CONTROL - an executive module whose prime function is to invoke other modules.

2 INPUT/OUTPUT - a module whose prime function is to communicate data between the computer and the user.

3 PRE/POSTPROCESSOR - a module whose prime function is to prepare data for or after the invocation of a computation or data management module.

4 ALGORITHM - a module whose prime function is computation.

[✓ 5] DATA MANAGEMENT - a module whose prime function is to control the flow of data within the computer.

6 SYSTEM - a module whose function is the scheduling of system resources for other modules.

VIII. CONSISTENCY

1. Does the design representation comply with established standards CS.1(1)	[✓ Y]
2. Do input/output references comply with established standards CS.1(3)	[✓ Y]
3. Do calling sequences comply with established standards CS.1(2)	[✓ Y]
4. Is error handling done according to established standards CS.1(4)	[✓ Y]
5. Are variable named according to established standards CS.2(2)	[✓ Y]

6 [✓ Y]

ETRIC WORKSHEET 2B
ESIGN/MODULE LEVEL

SYSTEM NAME: AMT
MODULE NAME: AMSPUT

Pg. 4

IX. INSPECTOR'S COMMENTS

Make any specific or general comments about the quality observed while applying this checklist?

ETRIC WORKSHEET 3
SOURCE CODE/MODULE LEVEL

SYSTEM NAME: AMT
MODULE NAME: AMSPUT

DATE: 1 July 91
INSPECTOR: McGinley

I. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY)

1. Number of lines of code MO.2(2) — 84
2. Number of lines excluding comments SI.4(2) — 36
3. Number of machine level language statements SD.3(1) — Ø
4. Number of declarative statements SI.4 — 7
5. Number of data manipulation statements SI.4 — 8
6. Number of statement labels SI.4(6) (Do not count format statements) — 1
7. Number of entrances into module SI.1(5) — 1
8. Number of exits from module SI.1(5) — 1
9. Maximum nesting level SI.4(7) — 2
10. Number of decision points (IF, WHILE, REPEAT, DO, CASE) SI.3 — 5
11. Number of sub-decision points SI.3 — Ø
12. Number of conditional branches (computed go to) SI.4(8) — 3
13. Number of unconditional branches (GOTO, ESCAPE) SI.4(9) — Ø
14. Number of loops (WHILE, DO) SI.4(3) — 2
15. Number of loops with jumps out of loop SI.4(3) — 1
16. Number of loop indicies that are modified SI.4(4) — Ø
17. Number of constructs that perform module modification (SWITCH, ALTER) SI.4(5) — Ø
18. Number of negative or complicated compound boolean expressions SI.4(2) — Ø
19. Is a structured language used SI.2 — [✓ Y]
20. Is flow top to bottom (are there any backward branching GOTOs) SI.4(1) — [✓ Y]

II. CONCISENESS (MAINTAINABILITY) - SEE SUPPLEMENT

1. Number of operators CO.1 — 1
2. Number of unique operators CO.1 — 1
3. Number of Operands CO.1 — 1
4. Number of unique operands CO.1 — 1

III. SELF-DESCRIPTIVENESS (MAINTAINABILITY, FLEXIBILITY, TESTABILITY, PORTABILITY, REUSABILIT

1. Number of lines of comments SD.1 — 48
2. Number of non-blank lines of comments SD.1 — 26
3. Are there prologue comments provided containing information about the function, author, version number, date, inputs, outputs, assumptions and limitations? SD.2(1) — [✓ Y] N
4. Is there a comment which indicates what itemized requirement is satisfied by this module? ~~SD.2(1)~~ TR.1 — Y [✓ N]
5. How many decision points and transfers of control are not commented? SD.2(3) — Ø
6. Is all machine language code commented? SD.2(4) — N/A
7. Are non-standard HOL statements commented? SD.2(5) — N/A
8. How many declared variables are not described by comments? SD.2(6) — Ø
9. Are variable names (mnemonics) descriptive of the physical or functional property they represent? SD.3(2) — [✓ Y]
10. Do the comments do more than repeat the operation? SD.2(7) — [✓ Y]
11. Is the code logically blocked and indented? SD.3(3) — [✓ Y]
12. Number of lines with more than 1 statement. SD.3(4) — Ø
13. Number of continuation lines — Ø

ETRIC WORKSHEET 3 SOURCE CODE/MODULE LEVEL	SYSTEM NAME: AMT MODULE NAME: AMSPUT	Pg. 2

IV. INPUT/OUTPUT (RELIABILITY, FLEXIBILITY, PORTABILITY)

1. Number of input statements MI.1(2) Ø
2. Number of output statements MI.1(2) 3
3. Is amount of input that can be handled parametric? GE.2(3) Y [✓ N]

4. Are inputs range-tested (for inputs via calling sequences, global data, and input statements) ET.2(2) Y
5. Are possible conflicts or illegal combinations in inputs checked? ET.2(3) Y
6. Is there a check to determine if all data is available prior to processing? ET.2(5) Y

V. REFERENCES (RELIABILITY, MAINTAINABILITY, TESTABILITY, FLEXIBILITY, PORTABILITY, REUSABILI

1. Number of calls to other modules MO.2(1) 1
2. Number of references to system library routines, utilities, or other system provided functions SS.1(1) 1
3. Number of calling sequence parameters MO.2(3) 1
4. How many elements in calling sequences are not parameters? MO.2(3) Ø
5. How many of the calling parameters (input) are control variables? MO.2(3) Ø

6. How many parameters passed to or from other modules are not defined in this module? MO.2(3) Ø
7. Is input data passed as parameter? MO.2(4) Y
8. Is output data passed back to calling module? MO.2(5) [✓ Y]
9. Is control returned to calling module? MO.2(6) [✓ Y]

VI. DATA (CORRECTNESS, RELIABILITY, MAINTAINABILITY, TESTABILITY)

1. Number of local variables SI.4(10) 1
2. Number of global variables SI.4(10) 10
3. Number of global variables renamed CS.2(3) SE.1(3) Ø

4. How many global variables are not used consistently with respect to units or type? CS.2(4) Ø
5. How many variables are used for more than one purpose? CS.2(3) Ø

VII. ERROR HANDLING - (RELIABILITY)

1. How many loop and multiple transfer index parameters are not range tested before use? ET.3(2) Ø
2. Are subscript values range tested before use? ET.3(3) [✓ Y] N
3. When an error condition occurs, is it passed to the calling module? ET.1(3) [✓ Y] N
4. Are the results of a computation checked before outputting or before

VIII. (EFFICIENCY)

1. Number of mix mode expressions? EE.3(3) Ø
2. How many variables are initialized when declared? EE.3(2) Ø
3. How many loops have non-loop dependent statements in them? EE.2(1) Ø
4. How many loops have bit/byte packing/unpacking? EE.2(5) SE.1(6) Ø
5. How many compound expressions defined more than once? EE.2(3) Ø

ETRIC WORKSHEET 3 OURCE CODE/MODULE LEVEL	SYSTEM NAME: AMT MODULE NAME: AMSPJT	Pg. 3

X. PORTABILITY

. Is code independent of word and character size? MI.1(3) [✓ Y] N

. Number of lines of machine language statements. MI.1 Ø

:. Is data representation machine independent? MI.1(4) [✓ Y] N

. Is data access/storage system software independent? SS.1 [✓ Y] N

X. FLEXIBILITY

1. Is module table driven EX.2(2) Y

2. Are there any limits to data values that can be processed? GE.2(4) [✓ Y]

3. Are there any limits to amounts of data that can be processed? GE.2(3) [✓ Y]

4. Are accuracy, convergence and timing attributes parametric? EX.2(1)

:I. DYNAMIC MEASUREMENTS (EFFICIENCY, RELIABILITY)

During execution are outputs within accuracy tolerances? AY.1(5) Y

N/A

2 During module/development testing, what was run time? EX.2(3)

3. What was budgeted run time?

Complete memory map for execution of this module SE.1(4)

		Size (words of memory)
4.	APPLICATION	
5.	SYSTEM	
6.	DATA	
7.	OTHER	

8. During execution how many data items were referenced but not modified EE.3(6)

9. During execution how many data items were modified EE.3(7)

XII. INSPECTORS COMMENTS

Make any general or specific comments that relate to the quality observed by you while applying this checklist:

WORKSHEET REPORT

The worksheet report displays the raw data entered in each worksheet. It represents the current values in the data base. It is used to verify and track data entry.

```
                         AUTOMATED MEASUREMENT TOOL
                              WORKSHEET REPORT
                                WORKSHEET 3

DATA Base Amtexs
MODULE: EXSGET                                      DATE: 12/23/81

I.  STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY)
    1.  NUMBER OF LINES OF CODE                                          95.
    2.  NUMBER OF LINES EXCLUDING COMMENTS                               47.
    3.  NUMBER OF MACHINE LEVEL LANGUAGE STATEMENTS                       0.
    4.  NUMBER OF DECLARATIVE STATEMENTS                                  4.
    5.  NUMBER OF DATA MANIPULATION STATEMENTS                            5.
    6.  NUMBER OF STATEMENT LABELS (EXCLUDING FORMAT STATEMENTS           0.
    7.  NUMBER OF ENTRANCES INTO MODULE                                   1.
       ENTER [CR] TO CONTINUE 'E' TO EXIT:


    8.  NUMBER OF EXISTS FROM MODULE                                      2.
    9.  MAXIMUM NESTING LEVEL                                             3.
    10. NUMBER OF DECISION POINTS (IF, WHILE, REPEAT, DO, CASE)          10.
    11. NUMBER OF SUB-DECISION POINTS                                     0.
    12. NUMBER OF CONDITIONAL BRANCHES (COMPUTED TO GO                    6.
    13. NUMBER OF UNCONDITIONAL BRANCHES (GOTO, ESCAPE)                   0.
    14. NUMBER OF LOOPS (WHILE, DO)                                       4.
    15. NUMBER OF LOOPS WITH JUMPS OUT OF LOOPS                           0.
    16. NUMBER OF LOOPS INDICIES THAT ARE MODIFIED                        0.
    17. NUMBER OF MODULE MODIFICATIONS (SWITH, ALTER)                     0.
    18. NUMBER OF NEGATIVE OR COMPLICATED COMPOUND BOOLEAN EXPRESSIONS    0.
    19. IS A STRUCTURED LANGUAGE USED?                                  YES
    20. IS FLOW TOP TO BOTTOM (ABSENSE OF BACKWARD BRANCHING GOTO's)?   YES

II. CONCISENESS (MAINTAINABILITY)
    1.  NUMBER OF OPERATORS                                               4.
    2.  NUMBER OF UNIQUE OPERATORS                                        1.
    3.  NUMBER OF OPERANDS                                                8.
    4.  NUMBER OF UNIQUE OPERANDS                                         3.
       ENTER (CR) TO CONTINUE, 'E' TO EXIT:
```

EXCEPTION REPORT

The exception report delivers the relationship of each module to a given threshold value of a particular metric. The relationship (less than, equal to, or greater then) and the threshold value is input from the user. This report can be used to identify modules whose scores do not meet a certain threshold, identifying them as potential problems.

```
                  AUTOMATED MEASUREMENT TOOL
                      EXCEPTIONS REPORT

DATABASE:  AMTEXS                                DATE: 12/23/81

METRIC: ET. 2
PHASE: MODULE IMPLEMENTATION
THRESHOLD VALUE: 0.65
RELATION: LESS THAN

         THE FOLLOWING MODULES ARE WITHIN RANGE REQUESTED

                  MODULE NAME          VALUE

                    EXSCEX             0.
                    EXCDLP             0.500
                    EXSDBG             0.333
                    EXSHLP             0.
                    EXSPGR             0.
                    EXSUPK             0.
```

NORMALIZATION REPORT

The Normalization Report provides the user with the overall rating of a selected quality factor. A series of regression equations are displayed which have been empirically derived from research. The current metric values are substituted in the equations and a rating for the selected quality factor is calculated. Regression, equations exist for the quality factors reliability, maintainability, portability, and flexibility only:

```
                         AUTOMATED MEASUREMENT TOOL
                       NORMALIZATION FUNCTION REPORT


DATABASE: AMTEXS
MODULE: EXSGET                        DATE: 12/23/81


DESIGN NORMALIZATION FUNCTION         IMPLEMENTATION NORMALIZATION FUNCTION

                          FACTOR:  PORTABILITY

# NO DESIGN NORMALIZATION FUNCTION    PORTABILITY = -1.7 + .19 (SD.1) +
FOR PORTABILITY FACTOR                    .76(SD.2) + 2.5(SD.3) + .64(MI.1)
                                          SD.1 = 0.426
                                          SD.2 = 0.857
                                          SD.3 = 1.000
                                          MI.1 = 0.972

                                      PORTABILITY = 2.154
```

METRIC REPORT

This report calculates the value of each metric catagorized by factor and by development phase. This report is used to determine a total picture of the project as measurements are taken.

AUTOMATED MEASUREMENT TOOL
METRIC REPORT/MODULE IMPLEMENTATION PHASE

DATABASE: AMTEXS
MODULE: EXSGET DATE: 12/23/81

FACTOR	CRITERIA	METRIC	VALUE
CORRECTNESS	Traceability	TR.1	1.000
	Completeness	CP.1	0.667
	Consistency/Procedure	CS.1	1.000
	Consistency/Data	CS.2	0.500
RELIABILITY	Consistency/Procedure	CS.1	1.000
	Consistency/Data	CS.2	0.500
	Accuracy	AY.1	1.000
	Error Tolerance/Control	ET.1	1.000
	Error Tolerance/Input Data	ET.2	1.000
	Error Tol./Computational Fail.	ET.3	0.
	Design Structure	SI.1	0.625
	Complexity	SI.3	0.100
	Code Simplicity	SI.4	0.722
MAINTAINABILITY	Consistency/procedure	CS.1	1.000
	Consistency/Data	CS.2	0.500
	Design Structure	SI.1	0.625
	Complexity	SI.3	0.100
	Code Simplicity	SI.4	0.722
	Modular Implementation	MO.2	0.750
	Quantity of Comments	SD.1	0.426
	Effectiveness of Comments	SD.2	0.857
	Conciseness	CO.1	1.000
TESTABILITY	Design Structure	SI.1	0.625
	Complexity	SI.3	0.100
	Code Simplicity	SI.4	0.722
	Modular Implementation	MO.2	0.750
	Quantity of Comments	SD.1	0.426
	Effectiveness of Comments	SD.2	0.857
	Descriptiveness of Impl. Lang.	SD.3	1.000
PORTABILITY	Modular Implementation	MO.2	0.750
	Quantity of Comments	SD.1	0.426

FACTOR	CRITERIA	METRIC	VALUE
	Effectiveness of Comments	SD.2	0.857
	Descriptiveness of Impl. Lang.	SD.3	1.000
	System Software/Independence	SS.1	0.500
	Machine Independence	MI.1	0.972
REUSABILITY	Modular Implementation	MO.2	0.750
	Generality/Implementation	GE.2	0.750
	Quantity of Comments	SD.1	0.426
	Effectiveness of Comments	SD.2	0.857
	Descriptiveness of Impl. Lang.	SD.3	1.000
	System Software/Independence	SS.1	0.500
	Machine Independence	MI.1	0.972
FLEXIBILITY	Modular Implementation	MO.2	0.750
	Generality/Implementation	GE.2	0.750
	Data Storage Expansion	EX.1	0.
	Computational Extensibility	EX.2	0.500
	Quantity of Comments	SD.1	0.426
	Effectiveness of Comments	SD.2	0.857
	Descriptiveness of Impl Lang.	SD.3	1.000
INTEROPERABILITY	Modular Implementation	MO.2	0.750
EFFICIENCY	Iterative Processing	EE.2	1.000
	Data Usage	EE.3	0.668

STATISTICS REPORT

The Statistics Report provides a profile of COBOL constructs for each module.

```
                         AUTOMATED MEASUREMENT TOOL
                              STATISTICS REPORT

DATABASE:  AMTEXS
MODULE:  EXSGET                                           DATE:  12/23/81

NUMBER OF LINES OF CODE                                                95.
NUMBER OF PERFORM STATEMENTS                                            4.
NUMBER OF EXTERNAL CALLS                                                0.
NUMBER OF EXECUTABLE STATEMENTS (PROCECURE DIVISION)                   43.
NUMBER OF COMMENTS                                                     48.
NUMBER OF DECLARATIONS (DATA DIVISION)                                  4.
NUMBER OF LABELS                                                        0.
NUMBER OF I/O REFERENCES                                                6.
NUMBER OF REDEFINES (EQUIVALENTS)                                       0.
NUMBER OF LEVEL 88 DATA ITEMS (LOCAL VARIABLES)                         1.
```

SUMMARY REPORT

The summary report provides a summary of the metric scores for all of the modules in the system.

```
                    AUTOMATED MEASUREMENT TOOL
                      METRIC SUMMARY REPORT

DATABASE:  AMTEXS                                  DATE:  12/23/81

                         MODULE:  EXSGET

AY.1 = 1.000       CO.1 = 1.000       CP.1 = 0.667       CS.1 = 1.000
CS.2 = 0.500       EE.2 = 1.000       EE.3 = 0.668       ET.1 = 1.000
ET.2 = 1.000       ET.3 = 0.          EX.1 = 0.          EX.2 = 0.500
GE.2 = 0.750       MI.1 = 0.972       MO.2 = 0.750       SD.1 = 0.426
SD.2 = 0.857       SD.3 = 1.000       SI.1 = 0.625       SI.3 = 0.100
SI.4 = 0.722       SS.1 = 0.500       TR.1 = 1.000
```

QUALITY GROWTH REPORT

When the user wishes to track the value of a particular metric over time, the Quality Growth Report will furnish a tabular display of the scores of a selected metric over the plhases of the project. This report is used to track a particular metric through a project to see how its value changes.

```
                    AUTOMATED MEASUREMENT TOOL
                      QUALITY GROWTH REPORT

DATABASE:  AMTEXS
MODULE:  EXSGET                                      DATE:  12/23/81

METRIC                  DETAILED           MODULE
                        DESIGN             IMPLEMENTATION

 ET.2                    0.750                 1.000
```

MATRIX REPORT

This report displays the average and standard deviations for all metric values modules. This report displays all of this information in a matrix form allowing the user to easily identify modules with metric scores that vary from the system average.

```
                         AUTOMATED MEASUREMENT TOOL
                               MATRIX REPORT

DATABASE:  AMTEXS
PAGE =      1                                          DATE:  12/23/81

MODULE NAME              AY.1     CO.1     CP.1     CS.1     CS.2     EE.2

EXSCEX                   0.       1.000    0.       0.       0.       1.000
EXSCHK                   1.000    1.000    0.667    1.000    0.500    1.000
EXSCLP                   1.000    1.000    0.667    1.000    0.500    1.000
EXSDBG                   0.       1.000    0.       0.       0.       0.
EXSGET                   1.000    1.000    0.667    1.000    0.500    1 000
EXSHLP                   0.       1.000    0.833    1.000    0.500    0.
EXSPGR                   0.       1.000    1.000    1.000    0.500    1.000
EXSQRY                   0.       1.000    0.667    1.000    0.500    0.
EXSSSM                   0.       1.000    1.000    1.000    0.500    0.
EXSUPK                   0.       1.000    0.625    1.000    0.500    1.000

AVERAGE =                0.300    0.900    0.550    0.700    0.350    0.500
STANDARD DEVIATION =     0.438    0.316    0.401    0.483    0.242    0.527
```

MODULE REPORT

This report displays the catalog of modules that have been entered into the database. It providss a status report on the database.

```
                    AUTOMATED MEASUREMENT TOOL
                          MODULES REPORT

DATABASE:  AMTEXS                                  DATE:  12/23/81

                    WS1  CONTAINS SOME NIL VALUES
                    WS2A CONTAINS SOME NIL VALUES

       THE FOLLOWING MODULES ARE PRESENTLY IN THE CURRENT DATABASE:

                    1.  EXSCEX **      2.  EXSCHK *
                    3.  EXSCLP *       4.  EXSDBG **
                    5.  EXSGET *       6.  EXSHLP *
                    7.  EXSPGR *       8.  EXSQRY *
                    9.  EXSSSM *      10.  EXSUPK *

             TOTAL NUMBER OF MODULES IN DATABASE IS 10.

NOTE:  *   INDICATES BOTH WS2B AND WS3 CONTAIN SOME NIL VALUES.
NOTE:  **  INDICATES WS2B CONTAINS SOME NIL VALUES.
```

APPENDIX B
CONVERSION OF AMT
FROM VAX 11/780 TO
HONEYWELL 6000

SECTION B-1
INTRODUCTION

When designing the AMT (Automated Measurement Tool), the portability of the IFTRAN source code was a major consideration. The AMT contract stipulated that a fully running version of the AMT be delivered on the Honeywell 6000 series computer (GCOS operating system) located at RADC (the Rome Air Development Center, Griffiss Air Force Base, New York). In order to provide us with more efficient computer access, it was decided to develop the software on a VAX 11/780 computer, at our General Electric Facility in Sunnyvale, California and then ship a tape containing the source code to RADC, Therefore, the VAX version had to be implemented in very standard code using system dependent functions only when absolutely necessary. Whichever system dependent functions were used would be modified after the code had been moved to the Honeywell. This appendix describes the coding techniques used to assure the AMT code was kept as system independent as possible and the differences between the VAX and Honeywell system dependent functions.

SECTION B-2
CODING STANDARDS

Table B2-1 describes the standards established. Table B2-2 identifies the differences in the system dependent functions between the two computer environments.

Table B2-1 Code Standardization

1. Only INTEGER and REAL data types used.

2. No INTEGER*n data types used.

3. No LOGICAL data types used.

4. No CHARACTER or CHARACTER*n data types used.

5. Character strings are stored in integer arrays, one character per array element. Unused array elements are filled with blanks.

6. Input/Output of "character" arrays is performed with the implied DO in combination with the alphanumeric (A) field Format descriptor.

   Example:
   ```
   WRITE(CRT,100)(DBNAME(I), I=1, 15)
   100 FORMAT ( 'DATABASE NAME' ,  15A1)
   ```

7. System dependent functions (mainly file handling functions) are isolated into subroutines. This way, modifications to those functions need only be made in one place.

Table B2-2 System Dependent Function Differences

1. Creating Files

VAX

```
OPEN (UNIT = n, NAME = filename, TYPE = 'NEW')
```

VAX FORTRAN allows filename to be an integer array, which is the way AMT stores character strings.

H6000

```
CALL CALLSS ("ACCESS CF, filename, size, type")
```

or

```
CALL CALLSS (  string  )
```

where string = "ACCESS CF, filename, size, type"

Honeywell FORTRAN has no direct method for creating files. The CALLSS routine allows any timeshare command to be given from an executing FORTRAN program (the timeshare command being a character string enclosed in quotes). In this case, the ACCESS subsystem is called to create a new file. Note that AMT stores filenames in integer arrays. Therefore, in order to call CALLSS, each character stored in the filename integer array must first be concatenated into the timeshare command character string. Concatenation is performed by the Honeywell CONCAT routine.

2. Opening Files

VAX

```
OPEN (UNIT = n, NAME = filename)
```

Table B2-2 System Dependent Function Differences (Cont.)

H6000

CALL ATTACH (unit, "filename;", etc.)

Note that filename must be a character string. Each character stored in an AMT filename integer array must first be concatenated into the filename character string. Also note that the filename character string must be terminated by a semicolon.

3. Closing Files

VAX

CLOSE (UNIT = n, DISPOSE = 'SAVE')

H6000

CALL DETACH (unit, etc.)

4. Determing if a File Currently Exists

VAX

CALL LOOK (unit, filename, blocks, return code)

After calling LOOK:
IF return code = 0, the file exists.
IF return code = 1, the file exists, but is currently open.
Any other return code, the file does not exist.

H6000

CALL ATTACH (unit, filename, status, etc.)
(See Opening Files)

After calling ATTACH:

If status = octal (4000 0000 0000) or
octal (4004 0000 0000)

the file exists.

IF status = octal (4037 0000 0000)

Table B2-2 System Dependent Function Differences (Cont.)

the file exists, but is currently open.

Any other status

the file does not exist.

5. Opening Random Access Files

VAX

OPEN normally

H6000

Before accessing random files a call to RANSIZ must be made to specify the record size of the file. The file is then opened normally.

6. Suppressing Carriage Return and Line Feed

VAX

End FORMAT statement with dollar signfield descriptor. The cursor will remain positioned at its current location for the next write.

```
FORMAT (5X, 15A1, $)
```

H6000

Place an ampersand as the first print character in FORMAT statement. The write will begin where the cursor was previously positioned.

```
FORMAT ( '&', F6.2)
```

7. PROGRAM Statement

VAX

Allows the PROGRAM statement as the first line of a FORTRAN program.

H6000

Does not recognize the PROGRAM statement.

SECTION B-3
VAX TO H6000 TRANSFER TAPE

Copying Files from the VAX to the Transfer Tape (see also Section B-4).

1. Create a file named TRANSFER.LST. Enter into TRANSFER.LST the name of each file to be transferred (one filename per line).

2. $ ALLOCATE MTAn:

3. Physically mount tape on drive #n.

4. $ INITIALIZE/DENSITY=1600 MTAn: label

5. $ MOUNT/DENSITY=1600/FOREIGN/BLOCK=80 MTAn: label

6. $ RUN TAPE2
   - When prompted for the output file name enter MTAn:
     (where n = number of tape drive allocated)
   - When prompted for the VAX file list enter TRANSFER.LST
   - At the end of its execution TAPE2 will display the message ALL FILES COPIED

7. $ DISMOUNT MTAn:

8. Physically remove the tape.

9. $ DEALLOCATE MTAn:
   The tape is now ready to be sent to RADC.

Reading the VAX Tape on RADC's H6000 (see also Section B-6)

1. Obtain the RADC tape number assigned to the transfer tape.

2. Edit file /AMTS/TRANSLATE/TMP, substituting the new tape number in the TAPE9 IN card.

3. Create a sequential file named /AMTS/VAXDATA of maximum size 1000. The transfer tape will be written to this file.

4.
```
*CARDIN
```

5.
```
*RUN /AMTS/TRANSLATE/TMP
```

6. When the job has finished running:
```
     *CONVERT /AMTS/VAXDATA = *:TRAIL
```

7.
```
*RESAVE /AMTS/VAXDATA
```
   /AMTS/VAXDATA now contains the information that was stored on the transfer tape.

SECTION B-4
VAX LISTINGS

1. Listing of VAX File TRANSFER.LST.

```
100  EXSCEX.IFT
200  EXSDBG.IFT
300  RGSCMM.IFT
400  UTLCRE.IFT
```

APPENDIX C
DBMS SURVEY

SECTION C-1
PURPOSE

One of the reauirements of the contract was to attempt to utilize a DBMS in our system design. It was anticipated that the use of a DBMS would increase the portability of the system as well as reduce the required effort to develop the system. Because these assumptions did not hold true for MDQS, the DBMS on the H6000/GCOS target environment, a DBMS survey was conducted.

Section 2 discusses the problems of using MDQS. It states why the initial version of the AMT will have to be implemented with its own built-in data management functions.

Section 3 examines the utility of using alternative DBMS's deemed most likely to be available in target environments. These DBMS's were examined and compared according to certain criteria.

SECTION C-2
THE PROBLEMS OF USING MDQS

The use of a DBMS was initially considered to be a good design choice. Storage and retrieval of the metric data could be performed by the DBMS. It has turned out that the Honeywell provided DBMS, MDQS, is inappropriate for the AMT application. Thus, while DBMS's in general could be considered for other versions of AMT, the initial version on the RADC H6180/GCOS system will have to be implemented with its own built-in data management functions.

The two major reasons for not using MDQS are: (1) we would not be able to use any of our existing software, and (2) the system would not be transportable. MDQS is a completely self-contained DBMS. It was developed strictly for business applications in which data is simply stored and retrieved with a minor amount of manipulation. The manipulation is done by internal procedures written using MDQS - provided constructs. Thus, all of AMT would have to be implemented within the framework of MDQS. This is practically impossible considering the complexity of the parsing and measuring algorithms that are part of AMT. In addition, none of the existing code that performs the parsing function (approximately 2000 lines of code) could be used.

More importantly, if the AMT was developed under the framework of MDQS it would have to be totally converted. This conversion would be necessary either for interfacing with another DBMS of for running on another system. Constrainment of the portability of this prototype software development is unacceptable. The trade-offs of using or not using MDQS are summarized below:

Using MDQS: Advantages

- o Query capability
- o Data management routines provided

Disadvantages

- Resulting system not portable to 370 or 11/70
- No existing software could be used.

Because of the net unfavorability of using MDQS, our approach has been tc develop some very basic data management functions based on a data base specifications. These data management function provide the core functions of a DBMS. The system dependencies are isolated in a few of these routines. They will have to be re-written when the system is transported to anothe system. This is a significant improvement in the degree of portability of th system.

SECTION C-3

ALTERNATIVE DBMS'S FOR CONSIDERATION IN FUTURE AMT VERSIONS

In order to examine the utility of alternative DBMS's, we did a survey. Table C3-1 gives an overview of the DBMS's we considered which have the facilities to run on the target environments hardware and operating systems. Tables C3-2 through C3-7 include a detailed analysis of only those DBMS's thought likely to best fit the target environments on overall criteria. MDQS is also included. Of these latter DBMS's only the following are capable of being called from FORTRAN:

- TOTAL
- IDMS
- MRDS
- MRI

Accordingly, selection from this subset of four DBMS's would contribute the most portabiltiy to future AMT versions in the target environments, all other things being equal.

Table C3-1

Data Base Management Systems

I. IBM 370/OS

A. Self-Contained Systems

1. ARAP-Data Retrieval System
(IBM 370/115 and up, OS/VS1, OS/VS2. Interfaces with OS Telecomm subroutines.)

2. Computer Corporation of America - Model 204
(IBM 370 under OS/MFT, OS/MVT).

3. Infodata Systems Inc. - INQUIRE
(IBM 370 interfaces with OS)

4. Mathematica Inc. - RAMIS
(IBM 370 under OS. Uses OS facilities for I/O, but relies on no other system, Dependent utlities. Dependent utilities, contains own sort logarithm. TP and Timesharing interfaces are available).

5. Meade Technology Corp. - DATA/CENTRAL
(IBM 370, Model 40/135 up. Operates under all versions of OS including virtual. Implementation on new machine requires 12-18 months).

6. MRI Systems Corporation - SYSTEM 2000
(IBM 370 OS VM/CMS, OS/1100).

7. National CSS- NOMAD
(IBM 370 OS)

8. TRW OIM II
(IBM 370 OS/VS)

B. CODASYL - Type Systems

1. Cullinane Corporation - IDMS
(IBM 370, all operating systems)

2. International Data Base Systems - SEED
(Written in FORTRAN so it may be used on any machine with a FORTRAN computer. CPU's include IBM 370).

C. HOL - Based Non - CODASYL SYSTEMS

1. Cincom Systems, Inc. - TOTAL
(IBM 370 OS)

2. IBM Corporations - IMS

   (IBM/VS runs on System 370 models 138, 145, 148, 15511, 158, 16511, 168 and 3033, OS/VS1 and OS/VS2).

3. Insyte Data Corp. - DATA COM/DB
   (IBM 370, OS)

4. Software Ag - ADABAS
   (IBM 370, OS MVT)

II. Honeywell 6180/GCOS

A. Self - Contained Systems

   1. MDQS

B. CODASYL - TYPE Systems

   1. Honeywell information Systems IDS II
      (Honeywell L6, L64, and 6000/L66 systems operating under GCOS batch or communications environment).

C. HOL - Based NON - CODASYL Systems

   1. Cincom Systems, Inc. TOTAL

      (Honeywell Level 62 GCOS; Level 66/6000 GCOS)

III. Honeywell 6180/MULTICS

A. Self - Contained

   1. Honeywell Multics Relational Data Store - MRDS
      (HIS Series 60/Level 68 Hardware).

IV. PDP 11/70 UNIX

A. Self - Contained Systems

   1. Bell LABS - INGRESS
      (PDP 11/70; UNIX)

B. CODSDYL - TYPE Systems

   1. Cullinane Corporation - IDMS -11

      (similar to IDMS; see IDMS chart)

      (PDP 11/70; IAS

Table C3-2

MRI

FUNCTION	PROPERTY	PARAMETERS
I. DATA BASE DEFINITION	A. Item Description	Self-contained DDL. Items defined in terms of unique name and number. Names up to 250 characters in length. Data Types: integer, decimal, character, text, data, money. Variable length character and text fields up to 250 characters long. Any number of user-specified indexed fields.
	B. Logical Structure	Tree structure. Data Base Definition allows 32 levels of Schema records containing schema items to model entries of data records containing data items. Data Bases may be linked using HOL interface to form logical networks.
	C. Physical Structure	Each data base is a stand-alone entity, comprised of separate physical files for schema, data, structures, and indexes. Data stored in user-determined fixed-length blocks in physical hierarchies system does own deblocking. File inversion on selected fields.
	D. Access Methods	BDAM, BSAM, BPAM, QSAM.
	E. Special Storage Techniques	Alphanumeric fields are variable length within fixed length logical record via separate Extended Field Table. Hierarchical structure is used to eliminate data redundancy. A variety of storage techniques (ring, tree, dense list, etc.) are used to optimize specific DBMS processes.
II. DATA BASE CREATION AND III REVISION		Initial data load may be run as a one time, incremental or transaction processing procedure. This loading can be accomplished with any combination of Programming Language Extension, Self-Containe Language and/or Self-Contained Utility. Schema may be modified using self-contained language. System automatically performs any internal restructuring required.
IV. DATA MANIPULATION	A. Selection Level	Selection is at the item level. Items may be identified by schema name or alias.
	B. Operators, Comparators, Logical Complexity	Variety of DML'S: PLEX, SCL, RW. Comprehensive selection capability in all DML's. Uses Boolean, Threshold, indexed/non-indexed, text search and positional techniques. Local and global updates with dynamic reuse of deleted space. SCL supports virtual data items.

Table C3-2

MRI (cont.)

FUNCTION	PROPERTY	PARAMETERS
	C. Reporting	Self-Contained Languages suitable for heuristic browsing, simple reporting, and complex, formal report generation are available. Both user-defined and formal default options are provided. Report requests may be formulated dynamically or executed from data base stored procedures. Sorts, arithmetic expression processing, logic structures, and system supplied function are also included.
V. USER INTERFACE	A. Manipulation Languages	English-like. Self-contained Query/Update language for single/multiple user interactive and batch processing.
	B. Mode of Interaction	Interactive or batch. All Self-Contained and HOL languages supported in both batch and interactive modes. Interactive support via MRI's TP 2000, CICS, INTERCOMM, TSO, and others.
	C. Error Messages	English-like text messages provided for self-contained language user, and diagnostic return codes and messages for host language programs. Centralized Messages and Codes Manual. Microfiche early warning for systems support personnel.
	D. Documentation	Modular documentation designed to satisfy the needs of each type of user. Structured top-down from concepts to language specifications to administration and support. Strong reliance on examples and usage guidelines.
VI. APPLICATION PROGRAMMING	A. HOL Interface	Interface available for assembly, COBOL, FORTRAN, and PL/1 languages, HOL interface includes a precompiler which transforms English-like commands embedded in the HOL code to DBMS call commands and the necessary parameter lists. The interface allows run-time HOL interaction with up to sixteen open data bases at any point in time.
	B. Subroutine Capabilities	Modular HOL programming is supported with DBMS processing available to main line and subordinate modules. SCL commands ("strings") may be stored with the data base definition and executed by entering the string name. Parameters may be passed at execution time. Strings may be called by other strings and both retrievals and update may be performed. External command files may be read. Calculation definitions may be stored as virtual items in the data base. Calculation definitions may be parametric.

Table C3-2

MRI (cont.)

FUNCTION	PROPERTY	PARAMETERS
	C. Special Operators	HOL support for dynamic subsets (LOCATE), network retrievals (LINK), sorts (ORDER BY), and automatic return code processing (FOR RC) are examples of special operators. Special operators in the SCL includes histograms system functions (SUM, AVERAGE, STD DEVIATION, COUNT, MINIMUM, MAXIMUM), and user defined calculations using the () + - * / symbols are examples.
	D. I/O Outside DMS	Any format output file from HOL interface Program. Report files created by Self-Contained Language and Report Writer. Unload to value string format provides capability to move data-base across hardware types.
	E. Auxiliary Storage	Intermediate results may be stored and manipulated in the self-contained report writer. Work areas, database table pages, sort and scratch files managed by DBA *tuneable Buffer Manager*.
VII. DATA BASE SECURITY, INTEGRITY AND ADMINISTRATION	A. Data Validation	Automatic checking on all fields based on data type. Further data checking may be performed by user HOL programs. Customized data validation via the user exit facility of the Universal Software Interface.
	B. File Protection	Security via passwords at item level. Authorization for retrieval, update and/or qualification optional. Security by data value at the hierarchical record level. User exits available for custom security checking.
	C. Surveillance	Two levels of logging (accounting and system usage) provide data suitable for surveillance needs.
	D. Failure Protection	Multiple recovery techniques which allow the DBA to trade logging overhead for recovery speed. Capabilities range from automatic rollback to self-contained dump/restore utilities. Recovery techniques can be specified for each individual data base and can be changed upon command.

Table C-[illegible]
IBM 370/OS

IDMS: CODASYL-TYPE DATA BASE MANAGEMENT SYSTEM

FUNCTION	PROPERTY	PARAMETERS
I. DATA BASE DEFINITION	A. Item Description	User - assigned names, Formats are those of supporting host language.
	B. Logical Structure	Network structure is through CODASYL'S set relationships. Several types of relationship possible.Membership may be mandatory or optional, manual or automatic. Linkage options allow one or bidirectional pointer chains and member to owner pointers.
	C. Physical Structure	User may specify storage area for occurrences of record type. Other options: System handles allocation and optimization of peripheral storage. DB administrators may assign DB portions to physical areas.
	E. Access Methods	Chained, direct, randomized, sequential, secondary index.
	F. Special Storage Techniques	Space management paging technique.
II DATA BASE CREATION AND III REVISION		Input via user application programs, or load utility. Modification of total DB descriptions (schema) can be handled either with a reload or restructure. Subschema can be modified at any time.
IV. DATA MANIPULATION	A. Selection Level	At record level by record identifier or placement relative to other records, or secondary indexes.
	B. Operators, Comparators, Logical Comolexity	Function of the host language and user programs.
	C. Reporting	Reporting done through user program. OLQ facility.
V. USER INTERFACE	A. Manipulation Language	COBOL, PLI, Assembler Macro, CALL, FORTRAN
	B. Mode of Interaction	CV option allows several DMS tasks to share same copy of system in a multitask environment. CV perfoms task monitoring and threading of DBMS calls. System includes monitor interface and a TP monitor. With monitor, each task can access any DB areas available for the user's declared usage mode. GCI ensures that more than one task does not update the same record at the same time. Multi-threading and multi-tasking central version. Intergrated DB/DC functions.

Table C3-3

IBM 370/OS

IDMS: CODASYL-TYPE DATA BASE (cont.

FUNCTION	PROPERTY	PARAMETERS
	C. Error Messages	Compilation errors listed with COBOL source statements. Error status is returned after DML statement execution for all host languages.
	D. Documentation	Standard user documentation.
VI. APPLICATION PROGRAMMING	A. Hol Interface	Through call statements.
	B. Subroutine Capabilities	Function of user's application program.
	C. Special Operations	Limited to those provided by host language.
	D. I/O Outside DMS	Done through user programs.
	E. Auxiliary Storage	To be provided by user on his program area.
VII DATA BASE SECURITY, INTEGRITY & ADMINISTRATION	A. Data Validation	Data item integrity is user's responsibility. Record placement is verified by program following user placement options. System provides data dictionary reports to DB administrator to document DB contents.
	B. File Protection	Access restrictions via subschema. Normal, protected or exclusive retrieval or update can be specified for each area. Record level lock for concurrent update and deadlock protection.
	C. Surveillance	Security dump provides DB copy and statistics of DB contents. Any part of dump may be reloaded using the security restore utility.
	D. Failure Protection	Restart/recovery utilities

TABLE C-4

HONEYWELL 6180/GCOS

TOTAL: HOL-BASED DATA BASE MANAGEMENT SYSTEM

FUNCTION	PROPERTY	PARAMETERS
I. DATA BASE DEFINITION	A. Item Description	User assigned names. Data formats are those of supporting host language.
	B. Logical Structure	Network multilist structure implemented via chains of bidirectional pointers linking variable entry records on the basis of relationships specified by user. DB elements include items, groups, records, files. Multiple linkage paths may be extended over several data bases. Each linkage path corresponds to a single entry file which provides pointers to the chains of the linkage path.
	C. Physical Structure	Records are fixed length, although several record formats on any given file. Single-entry records are accessed by a randomizing procedure using a key value. Variable entry records are then accessed following pointer chains. All data sets can also be accessed serially.
	D. Access Methods	Disk access is through BDAM and/or VSAM.
	E. Special Storage Techniques	Multiple files can share an I/O buffer as specified by user, but single and variable entry data sets may not share same I/O buffer. A linkage path may be specified as "primary" to optimize physical placement of records. TOTAL provides dynamic reallocation of space and optimization of synonym chains as well as user control parameters which optimize seek time.
II. DATA BASE CREATION		Input via user application programs or optional database administrator utilities.
III. DATA BASE REVISION		New records can be added, deleted, or modified from existing files. New data sets, linkage paths, record elements and modification of storage areas require DB regeneration, but not necessarily program or DB file modification.
IV. DATA MANIPULATION	A. Selection Level	At field level based on field values. Items described by name or by position (in the case of records) along with the linkage path.
	B. Operations, Comparators, Logical Complexity	Standard comparators. Complexity is function of user program.
	C. Reporting	Reporting via user programs with optional on-line query and batch reporting system capability. Output to all devices available to user programs.

Table C3-4 (Cont.)
HONEYWELL 6180/GCOS

TOTAL: HOL-BASED DATA BASE MANAGEMENT SYSTEM (cont.)

FUNCTION	PROPERTY	PARAMETERS
V. USER INTERFACE	A. Manipulation Language	Any language supporting subroutine calls.
	B. Mode of Interaction	TOTAL: Batch or on-line full multi-task, multi-thread system. Transaction logging (before and after images). Records locked (one per file per task) when task is updating. If locked record is requested by other tasks (as monitored by the system) it is released on a "time/request" algorithm to other tasks. Original task will be posted with a status indication.
	C. Error Messages	Error condition returned through user specified status variable when using data manipulation language.
	D. Documentation	Total DBA, total applications reference manual, total utilities, batch retrieval user guide, comprehensive retrieval user guide, on-line query user guide, data dictionary manual, data directory manual, on-line directory maintenance manual.
APPLICATION PROGRAMMING	A. HOL Interface	Data manipulation is via user application programs that issue calls to TOTAL.
	B. Subroutine Capabilities	Function of the supporting host language.
	C. Special Operations	Function of the supporting host language.
	D. I/O Outside DMS	Done through user-provided programs.
	E. Auxiliary Storage	To be provided by user on his program area.
VII. DATA BASE SECURITY INTEGRITY, & ADMINISTRATION	A. Data Validation	Structure validity provided by system. Additional integrity checking obtainable via special system exit to DB administrator programs.
	B. File Protection	Special exit is provided for interface with user-provided security procedures. Full DBA capabilities to control user access include sub-schema (logical view) which specifies user password, usable set of elements (data item names) and inter/intra file access.

Table C-4 (cont.)
HONEYWELL 6130/GCOS

TOTAL: HOL-BASED DATA BASE MANAGEMENT SYSTEM (cont.)

FUNCTION	PROPERTY	PARAMETERS
	C. Surveillance	Content validity must be assured by user application program.
	D. Failure Protection	Restart/recovery procedures are provided. Forward and backward processing of update history, optional automatic task level checkpoint, and other capabilities available under ENVIRON/1 (Cincom TP monitor), and CICS.

Table C3-5
IBM 370/CS

IMS: HOL-BASED NON-CODASYL DATA BASE MANAGEMENT SYSTEM

FUNCTION	PROPERTY	PARAMETERS
I. DATA BASE DEFINITION	A. Item Description	Items described in data base description (DBD) for DBMS sequencing and selection function, described in program at segment level for standard program use. No restrictions on types and coding in DBMS. With IMS/VS 1.1.5 will have field sensitivity.
	B. Logical Structure	Basic unit is segment but with 1.1.5 programs may retrieve, insert, replace, by fields. Also, logical structure may be obtained thru secondary indexing or logical relationships. Logical relationships may be between segments within the same physical database or different data bases. Structures may be inverted thru these logical relationships or secondary indexing.
	C. Physical Structure	Fixed length blocks. Variable length records and segments. Common buffer pool stores all data for DL/I language access.
	D. Access Methods	Access methods: HSAM, HISAM, HIDAM, HDAM. HSAM, HISAM are sequential. HDAM and HIDAM are direct. HDAM is randomized, HISAM, and HDAM support the inverted file and VSAM. VSAM can be used for HIDAM, HDAM, and HISAM data bases. Inverted data bases supported by all of above.
	E. Special Storage Techniques	Special storage techniques in HSAM, HISAM, HDAM, and HIDAM minimize storage requirements. For further data compaction an exit is provided in DL/I to a user routine. VSAM compacts indexes. Distributed free space can be requested at load or reorganization time to accommodate insertion of segments near their parents or twins. In HIDAM and HDAM, deleted segment space can be reused for new data.
II. DATA BASE CREATION		User program normally used for file creation.
III. DATA BASE REVISION		Logical structures are modified in the DBD and do not necessarily require file activity. Experience indicates minimal impact on programs. Physical structures are modified in the DBD and will normally require dumping and reloading of the DB.
IV. DATA MANIPULATION	A. Selection Level	At segment level. Items can be described by names, codes, or relationship to other items. Can be made by requesting a single segment, or a path of segments, or in 1.1.5 by retrieving by field. Any field in a segment can be used in a search argument.

IBM 370 DC
IMS: HOL-BASED NON-CODASYL DATA BASE MANAGEMENT SYSTEM (cont.)

FUNCTION	PROPERTY	PARAMETERS
	B. Operators, Comparators, Logical Complexity	Operators: AND, OR, LOGICAL AND. Comparators: EQ, NOT EQ; GT, GTEQ; LT, LTEQ. Limited, heuristic, and special structure searches. Eight logic combinations can occur at each segment level.
	C. Reporting	Reporting via host language or GIS. GIS produces default reports, page numbering and multiple page headers automatically. Output to all devices supported by IMS or the CPU. A response can be reviewed at a terminal and then sent to some other terminal for further processing/review.
V. USER INTERFACE	A. Manipulation Language	Through service calls from host language. English-like query language. (GIS)
	B. Mode of Interaction	Batch and on-line. Access lockouts at the page level for concurrent update purposes. Concurrent retrieval is always possible. IMS/DS provides interminal communications and remote job control with dynamic priority assignment. Programs are not locked out, they will always schedule into the message region(s) to process. Program isolation allows two or more programs to operate concurrently. If a user program updates a particular segment, no other program can access that segment until the update program reaches a synchronization point, or is complete. (Program Isolation in DC).
	C. Error Messages	Status code returned in response to all requests for data. User can check for error. Trace facility can be invoked at test time to provide data on each DL/1 call. Malfunctions and errors, displayed at the IMS master terminal.
	D. Documentation	DI/1 general information manual GH20-1260, terminal operator guide SH20-9028, system program reference manual SH20-9027, applications program reference manual SH20-9026, system applications design guide SH20-9025, utilities reference manual SH20-9029, messages and codes reference manual SH20-9030, IMS/VS conversion planning guide SH20-9034, systems documentation (licensed), message format service guide SH20-9052, and advanced function for communications SH20-9054. GIS general information manual GH20-9035, executive query reference guide GH20-9043, language reference manual SH20-9038, MSG and codes SH20-9039, advanced query reference manual SH20-9040, program reference manual SH20-9037, and systems documentation (licensed).

IBM 370/OS

IMS: HOL-BASED NON-CODASYL DATA BASE MANAGEMENT SYSTEM (cont.)

FUNCTION	PROPERTY	PARAMETERS
VI. APPLICATION PROGRAMMING	A. HOL Interface	Standard call interface specifying: function, logical file, I/O area, search argument. Implemented for COBOL, PL/1, ALC.
	B. Subroutine Capabilities	Standard host language rules apply.
	C. Special Operations	None.
	D. I/O Outside DMS	Selected sets become named files in three possible states: a vector file, an ordered list file, or the data file itself. Data file can be saved on any supported device via the DUMP command. Vector file or ordered list file savable in multithread version.
	E. Auxiliary Storage	GIS provides permanent and temporary files.
VII. DATA BASE SECURITY, INTEGRITY, & ADMINISTRATION	A. Data Validation	Checking only for data structure and sequencing. Exit provided for user program. Editing facilities in query language.
	B. File Protection	Segment sensitivity and processsing intent level of control-done in program specification block (PSB). User provided encryption, decryption can be implemented within the DMS through a special exit. Password and user profile carry security to the field level and beyond with qualification of user. Field sensitivity in IMS/VS 1.1.5 (and intent).
	C. Surveillance	All activities logged, including security violations. Logs available for statistical processing.
	D. Failure Protection	System automatically logs all changes to any data base and provides complete recovery utilities for restoring data bases without re-executing application programs. Checkpointing and restart facilities including synch of DL/1 and OS checkpoints and critical areas in the application program are also provided. System can continue running if application program fails.

Table C3-6
HONEYWELL 6180/GCOS
MRDS: SELF-CONTAINED DATA BASE MANAGEMENT SYSTEM

FUNCTION	PROPERTY	PARAMETERS
I. DATA BASE DEFINITION	A. Item Description	Naming: 1 to 32 character names. Format: Standard PL/1 data type declarations. String Types: Fixed/varing length bit and character strings. Arithmetic Types: Real or complex, fixed or floating, binary or decimal. Alignment: Word aligned, byte aligned, or unaligned.
	B. Logical Structure	Groupings: Data base, files, relations, tuples, attributes, domains. Linkage: See "Physical Structure." Structures: Relational. List, tree, network structures definable at query time.
	C. Physical Structure	Data Base: Implemented as a directory and subordinate files in the Multics Storage System. Disk Assignment: Interrelation clustering (optional), fixed and variable length fields. Ordering: Ascending primary keys. Linkage: Direct links, secondary indexes.
	D. Access Methods	System Interface: Multics virtual file manager (vfile–). No special I/O. Methods: Keyed sequential, random, linked, and/or hashed.
	E. Special Storage Techniques	Compaction: Encode and decode procedures. Variable length fields. Unaligned data. Efficiency: Interrelation clustering. Blocked (pre-allocated) files for hashing. Otherwise, keys are stored as B*-tree.
II. DATA BASE CREATION		Creation: "create–mrds–db" Multics command which translates a user written data model source and creates a corresponding data base shell. Loading: Applications program(s) or Linus EUF "store" request.
III DATA BASE REVISION		Utilities: 'restructure mrds db' Multics command allowing redefine, define, and undefine operations on files, relations, attributes, secondary keys, and foreign keys. Minimal to no impact on application programs using submodels.

Table C3-6 (Cont.)
HONEYWELL 6180/GCOS
MRDS: SELF-CONTAINED DATA BASE (cont).

FUNCTION	PROPERTY	PARAMETERS
IV. DATA MANIPULATION	A. Selection Level	Selection: attribute(s), tuple(s), relation(s). Qualification: attribute(s),or function(s) of attribute(s).
	B. Operators Comparators, Logical Complexity	Comparators:=, ^ =, >, <, > =, < =. Arithmetic Operators: +, -, *, / Builtin Scalar Operators: abs, after, before, ceil, concat, floor, index, mod, reverse, round, search, substr, verify. Built-inSet Operators: differ, inter, union, Boolean Operators: & ¦, ^ Other Operators: User definable scalar functions. Logical Complexity: Unrestricted. Relationally complete. *Linus EUF also includes the builtin set operators avg, count, max, min, sum, and user definable set functions.
	C. Reporting	Sorting: Interface to standard Multics sort commands. Reports: Interface to the Multics Report Program Generator (MRPG). *Linus EUF contains, in addition to the above, a basic report capability with controllable (or default) headers and column-widths, settable break-limits, and interfaces to the Multics File System and Lister facility.
V. USER INTERFACE	A. Manipulation Languages	HOL relational calculus selection expressions. Linus EUF: HOL Sequel-like selection expressions.
	B. Mode Of Interaction	Interactive, Absentee (batch), RJE; Interface at Multics command level, Linus EUF subsystem, or application program Call; Concurrency request thru r/d/s/m permission at data base or relation level.
	C. Error Messages	Creation: Compiler-like error messages at date base and data submodel creation time. Application Programs: Symbolic status/error codes translatable into short or long messages. EUF: Status/error messages within Linus.
	D. Documentation	MRDS Reference Manual (AW53). Linus Reference Manual (AZ49). MRPG Reference Manual (CC69). Multics "help" command and Linus "help" request. Marketing Education F31 and F32 course workbooks.

Table C3-6 (Cont.)
HONEYWELL 6180/GCOS

MRDS: SELF-CONTAINED DATA BASE (cont.)

FUNCTION	PROPERTY	PARAMETERS
VI. APPLICATION PROGRAMMING	A. Hol Interface	Languages: "call" interface from all Multics programming languages, Selection expression is passed as a character string argument. COBOL DML verbs also supported.
	B. Subroutine Capabilities	Full capability to store procedures, directly callable from command level and/or other programs passing arguments. Recursion and inter-language calls fully supported. Linus EUF has macro storing an invoking capability.
	C. Special Operators	MRDS automatically performs data conversions following ANSI PLI conversion rules. *Linus EUF has set operators avg, count, max, min, sum and arithmetic expressions which operate on the data after retrieval.
	D. I/O Outside DMS	Transportability: Data is completely transportable and/or directly usable by other Multics facilities such as the graphics system, text formatter, report writer, and application programs.
	E. Auxiliary Storage	Temporary Working Areas: Temporary relations which become a logical (and physical) extension to the data base for the user defining them. Permanent Working Areas: Standard Multics files.
VII DATA BASE SECURITY, INTEGRITY, & ADMINISTRATION	A. Data Validation	Validation: Domain verification enforcable at store and modify times. Integrity: Encode and decode normalization. Interrelation integrity enforceable via foreign key concept.
	B. File Protection	Level: Access rights definable at data base, file, relationa and attribute levels. Permissions: Retrieve, modify, store, delete permissions. Qualification: Person id, project id and/or global. Enforcement: Hardware and software enforcement via Multics Access Control List and/or ring mechanism.
	C. Surveillance	Within DBMS: None at the present time. Outside DBMS: Standard Multics facilities for auditing access violations.

Table C3-6
HONEYWELL 6180/GCOS
MRDS: SELF-CONTAINED DATA BASE (cont.)

FUNCTION	PROPERTY	PARAMETERS
	D. Failure Protection	Backup: Standard Multics backup and retrieve facilities "dump mrds db" command to backup (to tape) a quiescent data base. Rollback: Commitment/rollback capability (at file manager level) is cuurently under development. Restart: Standard Multics Emergency Shut-Down (ESD) and restart capability.

TABLE C3-7
HONEYWELL 6180/GCOS

MDQS: Self-Contained Data Base Management System

FUNCTION	PROPERTY	PARAMETERS
I. DATA BASE	A. Item Description	1) Data - Item identifiers can consist of a simple 30 - character name or it can consist of that name plus an entry - name qualification, a mask option, and or a conversion subroutine specification.
	B. Logical Structure	1) Elements: Data (field), record, file, data base; schema; networks and hierarchies. 2) The Application Definition File (ADF) is prepared by the data base administrator for the MDQS user. The ADF contains; data base reference name, entry names, and item names. 3) Relational items
	C. Physical Structure	1) The CREATE statement createsone or more new sequential or indexed - sequential data bases from one or more existing (transaction) data bases, with transformation of the transaction entries into the forms predefined for the desired new data base entries.
	D. Access Methods	1) Sequential, index sequential, and integrated 2) Concurrent data base access
	E. Special Storage Techniques	1) Implicit storing of the new - entry data base is performed only if the CREATE statement is unlabeled. 2) In explicit storing the user can specify a WHEN SEQUENCE error to do additional processing.
II. DATA BASE & CREATION AND III. MAINTENANCE		1) Data base creation and maintenance permits a user to: • access data with full concurrency • create a data base from one or more transaction data bases • update multiple data bases from multiple transaction files • write and or read auxiliary files ressiding on disk or tape • combine two or more data bases into a single data base • split a data base into two or more data bases • create a data base that is a subset of a data base
IV. DATA BASE MANIPULATION	A. Selection Level	1) At the element level for interactive users

Table C3-7

HONEYWELL 6180/GCOS

MDQS: SELF-CONTAINED DATA BASE (CONT.)

FUNCTION	PROPERTY	PARAMETERS
	B. Operators, Comparators, Logical Complexity	Five binary operators, unary operator, logical, relational. Full set of BOOLEAN operators. Conditional expression comparators.
	C. Reporting	Flexible parameters (or default specs) for page length, indenting, titles etc. Reporting is at the Query level. Defaults or user specifies titles, column separators, and data item display editing characters with clauses that serve as modifiers to the PRINT statement. These clauses are TITLE, COLUMN, and PRINT.
V. USER INTERFACE	A. Manipulation Language	The conversational Management Data Query (CMDQ) subsystem through a conversation with the terminal user, generates a MDQS procedure which will access a data base and display the desired information at the terminal or optionally on a file for later viewing. The Query Language allows a user to generate a report. Primarily procedure selection.
	B. Mode of Interaction	On-line and batch.
	C. Error Messages	The user is given a list of the valid responses.
	D. Documentation	Standard references.
VI. APPLICATION PROGRAMMING		No HOL interface.
VII. DATA BASE SECURITY, INTEGRITY, & ADMINISTRATION	A. Data Validation	Data value integrity is user's responsibility. The Application Definition File (ADF) is prepared by the DBA for the MDQS user. The ADF describes the names of elements & contents of the data base.
	B. File Protection	The user must previously obtain user profile subsystem (UPS) permission from the DBA before execution commnads PASSWORDS.
	C. Surveillance	N/A
	D. Failure Protection	Restart/recovery/rollback

APPENDIX D

TOOL SURVEY

SECTION D-1
PURPOSE

A survey of software tools on the candidate target environments was conducted during the first phase of the AMT contract. The purpose of the survey was to identify software tools that could be incorporated in the AMT. The survey was limited to the candidate environments because it was felt it was beyond the scope of this effort to transport tools from other environments. The criteria for selection of a tool for consideration for incorporation in the AMT were:

- o Applicability to software measurement (Did the tool provide any metric data?)
- o Portability of tool (Can the tool be used on different hardware configurations?)
- o Interoperability of the tool (How many modifications to the tool are necessary?)
- o Usability of the tool (How much effort is required to learn how to operate the tool? How much effort is there to preparing input and interpreting output that was tool-driven?)

The results of this RADC Tool Survey are presented in matrix form in paragraph 3. Background information and analysis of the state-of-the-art of software tools and their applicability to metric appear in paragraph 2, preceding the RADC Tools Survey. As a result of tis analysis, selective RADC tools that have compatible hardware/operating systems with the target environments are also included, in the matrix of paragraph 3. Finally, paragraph 4 describes the actual tools to be used in the AMT, what other tools were considered for use, or what tools were applied during its development.

SECTION D-2
CODING AND IMPLEMENTATION: METRICS APPLICABILITY

The origin of code inspection was structured programming and allied software engineering technologies of the early 1970's. The goal of automated static analysis/evaluation has been to automate the compliance with the techniques and make a search of program properties.

The program parameters are structure-based (program logical and data structure, naming conventions, documentation conventions, etc.), control/data flow based (avoidance of undue control complexity; assurance of well-definedness of variables, etc.), and interface based (assurance of correspondence between modules, subsystem, inter-system, etc). The anomaly-detecting metrics have to do with standards enforcement (deficiencies in source code), whereas the predictive metrics quantify the logic of design and implementation.

For example, the JOVIAL Automated Metric System (JAMS) is designed to collect structural information about JOVIAL programs. GE's Integrated Software Development System (ISDS) provides a capability to analyze other languages including FORTRAN PDL, IFTRAN, and PASCAL. A major subsystem of ISDS, the generalized parser (GNP), the grammar description language (GDL) and grammar tables, provides this capability and will be used in the AMT.

Symbolic evaluation of code has as its goal the "interpretation" of program behavior at the programming language level. Assumption must be made about the environment, the deterministic properties of the programming language behavior, and the outcome of symbolic execution results. On systems such as DISSECT or MACSYMA the user interactively chooses a path and performs symbolic interpretation of actions along the chosen path. The system then displays the "formulas" to the user. The user compares original and implemented formulas for equality. Differences between computed and actual formulas are mistakes. Special formula formatting methods are used to make these differences highly visible. Final control software is not yet available. Symbolic evaluation has good candidate potential for the accuracy metrics at the system level.

The final type of static analysis tools, proof of correctness, can be used at the system level, subsystem level, or the module level as assessments of different levels of correctness. The Failure of Proof Method (FPM), uses a mathematical approach to proving the correspondence between a program and its formal specification. The consistency metric is highly visible here.

Dynamic testing is achieved through system exercising of programs. Typical self-testing metrics for higher level language systems have been built on a experimental basis and include:

- Automatic specified percentage of program logical segment coverage in any one test; aggregated test coverage of close to 100%.

- Assistance in setting input values and evaluating output values.

- Some form of automated results comparison.

These dynamic test tools consist of two basic modules, an instrumentation module and an analyzer module. The source language program is submitted directly to the instrumentation module. Then the instrumentation module accepts the source program of the module under test and instruments it by inserting additional statements in the form of counters or sensors. The instrumented source file is compiled and executed. At this point an analyzer module produces a report documenting the behavior under the test during its execution.

Typical metric - like data reported are:

- Max and min values of variables.
- Number and percentage of subroutine calls executed.
- Measures of program complexity.
- Statement consistency checks.
- Program cross-references.
- Trace capability.
- Flagging of non-ANSI code.
- Logically impossible - path detection.
- Subroutine argument/parameter verification.
- Data range check.
- If statement trace.
- Branch trace.
- Subroutine/statement timing
- Min/max assignment values.
- First/last assignment values.
- Min/max DO Loop Control Variable.
- Final DO Loop Index Value.
- Final branch values.
- Statement, path, segment, module interface or flow execution freauencies
- Specific data associated with each executable source statement.
- Subroutine retrace capability, complete calling tree, reverse execution capability.
- Performance indices for modules and input data.

A list of dynamic tools would include: JAVS, CABS, FAVS, RXVP, FORTUNE, CIP, FORSAP, FETE, PROGFORT, PROGTIME, TPL, and TAP.

The goal of mutation analysis is to show that small changes in program are discovered by test data. Conversely, the test data must be strong enough to catch the significant errors. Relevance to error detection metrics is obvious.

The Pilot Mutation System (PIMS) has been applied to FORTRAN and COBOL pilot systems. Magnitude of the mutant error is classified as:

- o Program does not compute.
- o Program computes but does not run test data.
- o Program compiles, test run is satisfactory, and the program is either logically equivalent to the original or test data is not good enough.

Reliability analysis is still in its infancy. The goal is to determine whether all defects have been reliably removed by tests. Any error must be made known by some combination of inputs. Following this theoretical approach of examining all possible input combinations is prohibitive in terms of cost effectiveness and computer time/capacity. The Next Error Discovery Predition method fails because software reliability simply does not follow the probability laws of hardware reliability.

SECTION D-3
MATRIX OF SOFTWARE TOOLS

The matrix of software tools having potential metric applicability follows in Figure D3-1. It includes tools currently in use or planned for at RADC and additional non-RADC tools also worthy of consideration for AMT development or usage. Figue D3-2 illustrates the Software Tools Survey Sheet used to collect information about the target environment's software tools.

MATRIX OF SOFTWARE TOOLS HAVING METRIC APPLICABILITY

TOOL OR SYSTEM NAME	FUNCTIONS	SOURCE	H/W OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	CONTACT	METRIC COVERAGE
1. Tools Currently	In Use or Planned Use at RADC								
1. Software Science Analyzer	Accepts COBOL source as input & produces S/W science para meters	Purdue University	PDP 11/70, Purdue Mace & IAS	COBOL	COBOL	"Elements of Software Science", 1977	Planned		Halstead's Measure Co.1
2. JXX	Automated tool which will inforce & police established programming standard	Softech	IBM 360/370	JOVIAL J73	JOVIAL 73	MITRE Tech Report	Operational	Mr. Richard Motto RADC/ISIS	
3. Basic Statistics Collector	Performs static Analysis of BASIC programs (ie, counts statements, loop nesting, #characters/line, etc.)	RADC/ISIS	H6180, Multics	PL/I	PL/I, BASIC	Basic Statistic Collector RADC-TR-76-9	Operational for PL/I BASIC (planned)	Mr. Douglas White RADC/ISIS	
4. Comm Software Development Package	Assists in the production of programs for communications applications	CSC	H6180, Multics	JOVIAL J3	JOVIAL J73 PL/I		Operational		
5. Data & Analysis Center for Software (DACS)	Source of empirical data on S/W dev and maintenance for RADC	RADC/ISIS	H6180 GCOS				Operational	Mr. John Palaimo RADC/ISIS	
6. Debug & Test Microprogram Tools	a) Global Cross Reference Analyzer b) Control Flow Analyzer c) Data Flow Analzer d) Timing Analyzer e) Source Code Comparator	US. Air Force		N/A	N/A	Reliability Programming, RADC-TR-79-173 DOD Manual 4120,17M	Operational	Mr. Donald Roberts, RADC/ISIS	

Figure D3-1

MATRIX OF SOFTWARE TOOLS HAVING METRIC APPLICABILITY

TOOL OR SYSTEM NAME	FUNCTIONS	SOURCE	H/W OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	CONTACT	METRIC COVERAGE
6. (CONT)	f) Symbolic Executor g) Test Case Generator h) Microprogram Execution Monitor	U.S. Air Force		N/A	N/A	Reliability Programming, RADC-TR-79-173, DOD Manual 4120, 17M	Operational	Mr. Donald Roberts, RADC/ISIS	
7. JOVIAL/J3 Statistics Collector	Counts, averages, & % of source program by type of data used size & nature of arrays, types of statements, comments DEFINE directives	U.S. Air Force	HONEYWELL 600/6000 GCOS	JOVIAL/J3	JOVIAL/J3	JOVIAL/J3 Statistics Collector, RADC-TR-77-293	Operational	Mr. Richard Slavinski RADC/ISIS	
8. JOVIAL Automated Verification System (JAVS)	Recognition of untested program paths, develop additional test cases, document the computer program with metric - like counters and sensors	U.S. Air Force	H6180 GCOS	JOVIAL	JOVIAL J3	JAVS Final Report RADC-TR-78-247; JAVS Technical Report RADC-TR-77-126	Operational	Mr. Frank La Monica, RADC/ISIS	
9. JOVIAL Compiler Validation Systems (JCVS)	Test Compiler	U.S. Air Force		JOVIAL J73	JOVIAL J73	JCVS RADC-TR-74-232; JCVS User's Guide RADC-TR-73-268	Being Developed	Mr. R.T. Slavinski, RADC/ISIS	
10. FORTRAN Automated Verification System (FAVS)	Static detection of unreachable statements, set/use errors, mode conversion errors, & external references errors; test case development & source code instrumentation automated documentation	U.S. Air Force	H6180 GCOS,	FORTRAN	DMATRAN	FAVS RADC-TR-78-268	Operational	Mr. Frank La Monica RADC/ISIS	

Figure D3-1 (Continued)

MATRIX OF SOFTWARE TOOLS HAVING METRIC APPLICABILITY

TOOL OR SYSTEM NAME	FUNCTIONS	SOURCE	H/W OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	CONTACT	METRIC COVERAGE
11. FORTRAN Code Auditor	Automated documentation, format, design, & structural standards	U.S. Air Force	H6180	FORTRAN	FORTRAN	FORTRAN Code Auditor, RADC-TR-76-395	Operational	Mr. Frank La Monica RADC/ISIS	
12. Semantics Oriented Language (SEMANOL)	Applied to proposed or implemented higher order language & will detect any consistencies in the HOL specifications	U.S. Air Force	HIS 600/6000 Multics	SEMANOL	JOVIAL/J3, JOVIAL/J73, CMS - 2, BASIC	SEMANOL RADC-TR-75-211; Improvements to SEMANOL RADC-TR-77-365	Operational	Mr. Douglas White RADC/ISIS	
13. AVIONICS Software Reliability Prediction Model	Prediction of the reliability and mean time to failure of the software development project	U.S. Air Force	H6180 Multics			Final document not yet published	Operational	Mr. Alan Sukert, RADC/ISIS	
14. WWMCCS Program Support Library (PSL)	PSL provides support & records of all aspects of the program development process including design, coding, testing, documentation, & maintenance	U.S. Air Force	HONEYWELL H6000/WWMCCS	COBOL	ANSI COBOL and GMAP		Operational	Mr. Lawrence Lombardo, RADC/ISIS	
15. COBOL Structured Programming Precompiler	Additions to COBOL, X .3.23-1968, are in the form of structuring verbs which permit the programmer to write the basic control logic figures required to implement structured programming forms.	U.S. Air Force	IBM 370 & HONEYWELL H6180	COBOL	COBOL	Structured Programming Series RADC-TR-74-300	Operational	Mr. Frank La Monica, RADC/ISIS	

Figure D3-1 (Continued)

TOOL OR SYSTEM NAME	FUNCTIONS	SOURCE	H/W OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	CONTACT	METRIC COVERAGE
15. COBOL Automated Verification System (CAVS)	Integrated, automated Test	U.S. Air Force	IBM 370	COBOL	COBOL		Planned	Mr. Frank La Monica RADC/ISIS	
II. Additional Software Tools to Consider Currently Not Listed as Being Used at the Target AMT Facilities									
1. Problem Statement Language/ Problem Statement Analyzer (PSL/PSA)	requirements documentation & analysis	Univ. of Michigan	IBM 370 H 6000 CDC 6000 V 1100	Program Specification Language	FORTRAN	PSL/PSA User's Guide	Operational; available	D. Teichroew Univ. of Michigan	
2. JAMS	JOVIAL code analysis	GE	PDP 11/40 RSX11M	JOVIAL 34	FORTRAN	IR&D Final Report	prototype	Jim McCall GE	
3. DISSECT & MASYMA	symbolic code analysis	Univ. of Massachusetts Univ. of CA, San Diego				IEEE Trans. Software/77 Engineering	Research aid	L. Clark, Univ. of Mass., W. Howden, Univ. of CA.	
4. Proof of Correctness Method (PCM)	mathematical approach to proving the correspondence between a program and its formal specification					IEEE Trans. Software Engineering 9/76	Research aid		

Figure D3-1 (Continued)

TOOL OR SYSTEM NAME	FUNCTIONS	SOURCE	H/W OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	CONTACT	METRIC COVERAGE
5. DYNAMIC TESTING OF CODE									
a. RXVP	Integrated, automated testbed to gather statistics of the program code.	GRC	H6000	FORTRAN	IFTRAN	GRC Manuals	operational proprietary	GRC	
b. FORTUNE		CAPEX	IBM 360/OS	FORTRAN	FORTRAN		operational		
c. CIP		NARDAC SYSTEMS SUPPORT	Honeywell 6000	COBOL	COBOL	User Manual NARDAC, Washington, D.C., #88S002B,TN02	operational	Naval Data Automation Command	
d. FORDAP&APAPS		UCLA	IBM 360/OS	FORTRAN	FORTRAN		research aid		
e. FETE		STANFORD	IBM 360/OS	FORTRAN	FORTRAN		research aid		
f. PROGFORT		STANFORD	IBM 360/OS	FORTRAN	FORTRAN		research aid		
g. PROGTIME		STANFORD	IBM 360/OS	FORTRAN	FORTRAN		research aid		
h. TPL		GE-CR&D	PDP-11/45 RSX-11D H6000; GCOS III CDC 6400	FORTRAN	FORTRAN	Test Procedures D.T. Panzl	operational	D. Panzl CRD, GE	
i. IAP		GRC	H6000	FORTRAN	IFTRAN		operational		
6. Pilot Mutation System (PIMS)	Makes small mod'fications (mucations) in original program to produce a mutant, then classify type of result. Error seeding	Univ. of Calif., Berkeley; Georgia Institue of Technology				Georgia Institue of Technology, Report GIT-ICS-79/08	prototype	R.J. Lipton University of Calif; Berkeley, Ca	

Figure D3-1 (Continued)

TOOL OR SYSTEM NAME	FUNCTIONS	SOURCE	H/W OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	CONTACT	METRIC COVERAGE
(CONT)									
7. Integrated Software Development System (ISDS)	Provides structural information about program code	GE	PDP11, 28K DOS/BATCH, RSX-11D VERSATEC P/P TEKTRONIX 4012	FORTRAN, PDL IFTRAN, PASCAL	IFTRAN/ Available in FORTRAN	TIS76CIS01	operational	Gene Walters GE, Sunnyvale	

Figure D3-1 (Continued)

GENERAL ELECTRIC

M&DSO WEST

SOFTWARE TOOLS SURVEY

TOOL OR SYSTEM NAME:	EXISTING OR PLANNED: E P DATE OPERATIONAL:
DEVELOPER: CONTRACT NO. (IF APPLICABLE):	IN PUBLIC DOMAIN OR PROPRIETARY?: PD P IF PROPRIETARY, APPROXIMATE COST:
IMPLEMENTATION LANGUAGE(S): IS SOURCE CODE AVAILABLE? Y N	TARGET LANGUAGE(S):

OPERATIONAL ENVIRONMENT

HARDWARE:

OPERATING SYSTEM:

COMPILERS:

SPECIAL REQUIREMENTS:
(e.g., a DBMS, graphics package, etc.)

FUNCTIONAL DESCRIPTION:

REFERENCES:

USAGE DESCRIPTION: (include users, results, and any references which describe usage results)

Figure D3-2

SECTION D-4
TOOLS USED

The tools used in AMT, considered for use, and applied during the development of the AMT are identified in Table D4-1. The tools identified as used in the AMT were actually incorporated in the software as part of the system. The tools identified as considered for use are candidates for interfacing with the AMT. This was not done because these systems were not available during the span of the project. The last category of tools identified are those tools used on the AMT, ie. these tools were utilized by the development team during the development of AMT. SPDL is a program design language with ADA-like constructs and concepts. The design was written in this language and some metrics automatically applied by the Integrated Software Development System (ISDS). The implementation language utilized was IFTRAN, a structured FORTRAN preprocessor developed by General Research Corporation.

Table D4-1
Tool Usage

TOOL	DESCRIPTION OF TOOL
<u>10 BE USED IN AMT</u>:	
- ISDS GNP	Generalized Parser - for structural analysis
- ISDS UTLCMP	Compare strings of characters
- H6000 GCOS UTLGTC	Obtain user enter command
<u>CONSIDERED FOR USE IN AMT</u>:	
- Software Science Analyzer (Purdue)	From COBOL source produces software science parameters.
- COBOL Usage Analyzer (Texas A&M)	Software tool for measuring usage of COBOL language by program.
- CAVS (GRC)	COBOL Automated Verification System
- COBOL Structured Pre-compiler	Structuring verbs allow for the implementation of structured programming forms.
- PSL/PSA	Problem Statement Language/Problem Statement Analyzer
<u>USED ON AMT</u>:	
- SPDL (GE)	Structured Program Design Language
- IFTRAN (GRC)	Structured FORTRAN
- ISDS (GE)	Integrated Software Development System

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.